IsoTex: Creating Isotypes as Documents with a Declarative Markup Language

ANONYMOUS AUTHOR(S)
SUBMISSION ID: 7081

```
\doc[orientation=portrait]  {

\character[alias=icon, weight=10, width=30px] {tree.png}
\character[alias=image, weight=1, width=150px] {earth.png}
\character[alias=text, color=#333333, width=11px]{Arial.ttf}

\sentence[characters=image]{1}
\sentence[characters=text]{= 10 billion trees}

\paragraph[width=30 character, vspace=10px]  {
     %3040 billions trees in the world
     \sentence[characters=icon]{3040}}
}
```

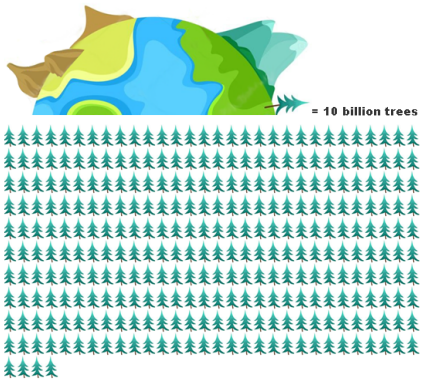


Fig. 1. Creating an isotype representation with the IsoTex language

This paper contributes IsoTex, a new semi-declarative markup language for creating isotypes. Isotypes are an omnipresent form of visualization favored by designers to communicate data to a diverse audience via series of pictorial representations or icons. IsoTex makes isotype creation akin to textual document preparation in LaTeX. Coupled with an interactive editor, IsoTex enables designers to explore a large range of design alternatives via either live editing or direct manipulation. Users can experiment with different data units and layouts in the language, immediately reflected in the editor, or rapidly skim through alternative icons, adjusting their aspect ratio and spacing in the editor, immediately added to the syntax. The light-weight version history supports design experimentation while the textual specification facilitates design reuse. Feedback from users suggests that IsoTex could facilitate design explorations compared to graphical editors while offering a lower learning curve than programming.

CCS Concepts: • **Human-centered computing → Human computer interaction (HCI)**.

Additional Key Words and Phrases: isotypes, declarative markup language, visualizations

ACM Reference Format:

Anonymous Author(s). 2018. IsoTex: Creating Isotypes as Documents with a Declarative Markup Language . In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/1122445.1122456

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
Manuscript submitted to ACM

1 INTRODUCTION

Isotypes are a family of visualizations where data units are encoded using series of icons—graphical representation of a concept—or pictograms—graphical representation of a physical object—organized in space (Figure 1) [20]. They are also called pictographs. The simplicity of their decoding, coupled with the expressiveness afforded by the icons make isotypes a widely popular representation for storytelling [19]. Designers and visualization practitioners have favored such representation for many years and they are omnipresent in infographics[1], data videos[2] [2] and news articles[3]. They also are one of the first visualizations taught at elementary school [1] as they facilitate the learning of complex visual-data mappings: from concrete visual representation with trivial mapping (e.g. using a physical apple to encode 1 apple) to a more abstract representation with complex mappings (e.g. using a drawing of a bag to encode 10 apples).

Crafting isotypes is deceptively simple as it involves only three design decisions: composing a palette of icons, defining units of data encoded by each, and specifying a layout. Yet, these decisions are tightly interlinked, making it difficult to predict the final result. For these reasons, experimenting with variations — and thus exploration of the design space for a given dataset — is a critical part of the creation process.

Since isotypes are both common in the visual communication of data, and they appear easy to create compared to other data visualizations, they appeal to a large audience of creators: from designers and professional storytellers, to a more general audience such as information workers, decision makers, or social media influencers. These creators are rarely well-versed in programming and use vector drawing tools that enable quick iteration of the visual imagery, or more generic presentation tools that enable rapid access to icon libraries and simple layout capabilities.

While such tools make the creation of a first version of an isotype achievable with reasonable efforts, iterating and exploring the design space (i.e., changing data units, visuals, or layout) is tedious and prone to errors, as the visual-data mapping is either done manually or in a separate tool. Programming alternatives such as Vega [11] or Altair [22] provide data binding, making it easier to experiment with data units. However, they require back-and-forth between graphics and programming environments [3], and they represent a steep learning curve to which few may be ready to commit.

We present IsoTex, a declarative markup language describing isotypes as documents, coupled with a live visual editor affording direct manipulation of the content. IsoTex is designed to reduce the gap between a design "mental model" and a programming "mental model". Inspired by LaTeX, IsoTex is a language describing isotypes using the terminology of textual documents (title, caption, paragraph, sentence, word, and characters) as it universally understood. The editor reinforces the link between the language specification and the visual output by highlighting portions of the language when hovering over graphical elements, which may facilitate learning. The editor also supports direct manipulation familiar to a wide range of users (e.g. drag a box border to resize its content), which results in direct updates of the language description of the isotype. Together, the language and editor provide a solution for a diverse audience to more easily craft isotypes, allowing rapid exploration of their design space.

Qualitative feedback from five users with diverse backgrounds (designer, data scientist, visualization practitioner, communication professional, and web developer) suggests that IsoTex can support a variety of workflows, enabling people to take advantage of data binding and fine parameter-tuning while not *"liv[ing] in the code"* and feeling *"disconnect[ed] from the visuals"* (Designer). We also demonstrate the expressivity of the IsoTex language via a gallery of examples available at http://isotex.github.io.

[1] https://www.cdc.gov/ncbddd/disabilityandhealth/materials/infographic-disabilities-ethnicity-race.html
[2] http://www.fallen.io/ww2/
[3] https://www.washingtonpost.com/graphics/national/police-shootings/

2 RELATED WORK

The goal of IsoTex is to enable designers and non-programmers to generate istoype visualizations. These representations pioneered by Otto Neurath [18] are particularly memorable [4, 5, 10, 16] and thus compelling for data driven storytelling. We designed the language and its editor as a middle ground between visualization authoring languages and GUIs.

2.1 Visualization Authoring Languages

Programming Languages. A substantial number of programming-based toolkits exists to create data visualizations. The most popular are probably D3 [6] and matplotlib [13] for JavaScript and Python respectively. These allow the authors to procedurally specify the position of arbitrary marks in a visualization or use pre-built routines to specify different types of visualizations. Users of these languages need to be fairly experienced programmers to write code and generate any visualization. While existing sample code may be adapted to achieve a visualization, iterating over different nuances of layout and icon appearance is often achieved in a graphical editor. In contrast to such programming languages, enabling to create a wide range of standard visualizations, IsoTex is tailored to a single family of visualization: isotypes. This focus enables to design a simpler and more succinct language.

Markup Languages. Markup languages are generally easier for people to understand as they do not require the same shift of mental model that programming does with procedures. Rather, it is similar to annotating portion of a text by inserting semantic tags and letting the system handle all of the formatting. The omnipresent markup language on the web is HTML and it is straightforward to see how the use of repeated IMG tags and line breaks would allow the construction of a simple isotype. This, when coupled with the styling available in CSS allows authors to modify these icons or color them differently. This approach however lacks any sort of data binding, which makes it error prone since images must be replicated manually and thus it is difficult to iterate on the designs. Our approach is similar in spirit, but allows for data-binding and semantic elements more closely aligned with the parts of an isotype. Several markup languages enable the embedding of data visualizations components such as Idyll [8] or Tangle [23]. Iterating over these components generated in external libraries poses the same barrier as learning programming languages for their users.

Grammars and Declarative Languages. Closer to our approach, researchers have created grammars for languages to specify visualizations in a declarative manner (ggplot [25], Vega-Lite [21], Altair [22]). The key difference with our proposal is that these languages use a data visualization terminology (e.g. visual marks, axis) that needs to be learned whereas we are building upon more generally understood document terminology (e.g. character, sentence).

2.2 Visualization Authoring Tools

The visualization community has also developed a number of tools that enable people to create visualizations without typing any code or textual specification.

Creating Isotypes as Textures. It is possible to create isotypes in mainstream data analysis and visualization tools such as Tableau or Excel by mapping a visual texture to bars in a bar chart. However, this texture needs to be manually created as an image in a different tool. Microsoft PowerBI has a dedicated extension [24] enabling the user to compose such textures from multiple icons. Yet, the data-visual binding is based on specifying the number of icons and their layout to fill a bar in a bar chart. It does not allow users to define a data unit and thus produce the corresponding icons. Data-driven guides [15] is a more advanced example of texture-based data binding. While these approaches reduce manual tuning, they make it difficult to explore the design space of data units.

Creating Isotypes with data-visual bindings. DataInk [26] enables authors to draw a sketched icon and bind it to each row of a data table. Comparing isotypes resulting from different data units thus requires manual labor — generating different data files and uploading them in the tool separately. DataIllustrator [17] and DataQuilt [27] enable authors to bind an icon to a visual element in a visualization. Similarly to DataInk, these tools do not offer a direct (and easily modifiable) binding between an icon and a unit of data other then the value one (one icon per data element).

2.3 Output-Directed Programming

Most IDEs such as jsfiddle (jsfiddle.net) or vscode (https://code.visualstudio.com/), offer a language or specification view and the corresponding output or result view generated by compiling or interpreting the language. Techniques to connect the specification to its output facilitate the development process [9]. An example of popular IDE for document markup is Overleaf, providing a LaTeX and a PDF view, as well as mechanisms coordinating navigation between the two.

However, in most IDEs, the editing flow is unidirectional: changes made by the user to the specification are then reflected on the output. A few tools enable bidirectionally, enabling the user to also interact with the output view to modify or create corresponding lines in the specification. Users can then work from either panel. The workflow enabled by such tools has recently been coined *Output-Directed Programming* [12]. Arguably, one of the first instances of output-direct programming is VisualBasic. Perhaps the most related to this present work is Sketch-n-Sketch [12], enabling output-direct programming to craft SVGs.

Output-Directed Programming lowers the barrier to developing programs and aids in learning how to code. Indeed, users may able to build the entire interfaces without typing a single line of code. Brushing and linking between visual components and their specifications help understand the connection. Generating portions of code also acts as samples and facilitates copy and reuse. This principle was recently applied in the context of computational notebook [14] to alternate between data analysis and standard visualizations. This present research applies output-directed programming to isotypes in the context of data-driven storytelling.

3 ISOTEX LANGUAGE

Our goal with IsoTex is to make the specification of isotypes easily understandable by non-programmers. While markup languages are more human-readable than programming languages by their declarative nature, they still require authors to understand a series of tags, what they mean, and how they can be combined together into a consistent syntax. The key idea of IsoTex is to rely on well understood terms for describing textual documents — namely paragraphs, sentences, and characters — in order to describe the entire family of isotype visualizations.

We first describe how to specify the general structure of an isotype (Figure 2). Then we describe how the isotype is built from the data, by defining characters and specifying the structure of a sentence (Figure 3).

3.1 General structure

At the highest level, a document contains the entire isotype. Parameters reflect what can be found in textual documents including width, height, background color or image, as well as the orientation and number of columns.

Documents contain **sentences** made of **characters** grouped into **words**. **Paragraphs** contain sentences divided into groups, enabling the creation of a higher level structure (e.g. rows or columns). Note that there are multiple ways to describe the example presented in Figure 3. In this example, the creator decided to build a single paragraph made of three sentences with a particular structure. However, an alternative strategy would be to have this document represented as three paragraphs, each containing a sentence with a year, followed by one with the icons.

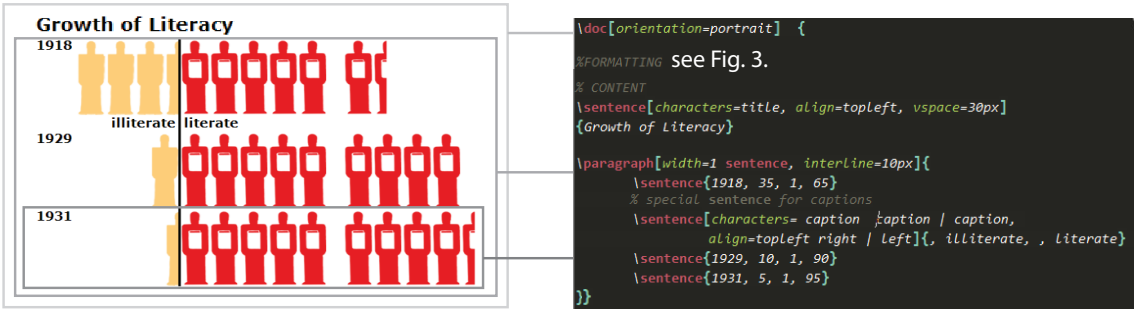


Fig. 2. An isotype contrasting the number of literate and illiterate people over three years and its description in IsoTex.

Fig. 3. Formatting of the content described in Figure 2 describing icon-data binding (1) and layout (2).

A key principle is that the content of sentences are either raw text (e.g. labels) or quantities of data to be represented by a series of icons. The unit of data encoded by each icon is specified at the character level, enabling the creator to easily explore multiple units and their effect on the resulting isotype.

3.2 Data binding

Characters and fonts enable the creator to define the visual vocabulary composing an isotype: icons, textual elements (e.g. labels, title), as well as visual separators (e.g. the black vertical line separating orange and red icons in Figure 2). Note that characters have a parameter `[alias]` to enable users to quickly refer to them later, when specifying their layout in the document. **Fonts** enable a higher level styling, applying a number of parameters to the set of characters they apply to.

The parameter `[weight]` is central in IsoTex as it enables *data binding* between the data and characters. In Figure 2, characters I and L have a weight of 10 (Figure 3 (1)), specifying how much data one icon represents. Thus, the sentence `\sentence {30}` with one of these characters will result in a line composed of 3 icons.

It is important to note that isotypes are a family of visualizations built for enabling their viewers to rapidly estimate and compare quantities by comparing a relatively small number of icons. Therefore, selecting the data unit to be most meaningful to the viewer and enable rapid additions is critical. For example it is better for 1 icon to represent 10 thousand people, than for it to represent 9.5 thousand. The creator must also select a data unit that will result in a small enough number of icons for each data quantity that they can be easily estimated, yet large enough that data quantities represented can be dissociated. The parameter `[weight]` is thus central for exploring the design space.

A data unit striking a balance between rapid visual estimation of icons and rapid computation of data quantity may make it impossible to depict differences between two quantities smaller than a whole data unit. When the problem occurs for a few quantities in the chart, designers may opt to fragment icons, to gain more precision, rather than opt to select a different unit altogether. The parameter `[clip]` (Figure 4 left) specifies if and how fragmentation is done.

As noted above, selecting a data unit for an icon has major impact on the overall readability of an isotype. Creators may opt for a meaningful and round unit of data at the expense of a large quantity of icons. They may then group these icons together to enable a more rapid estimation of their quantities. The parameter `[word]` (Figure 4 right) enables to specify such grouping, making the parallel with words separated by spaces in a sentence.

Fig. 4. Example of `[clip]` and `[word]`

3.3 Layout

Sentences enable the creator to specify the *layout* of each line of the isotype. Two major parameters are required for layout: the pattern of the visual elements (labels, icons and separators) and their positions relative to each other.

The parameter `[characters]` specifies the order of appearance of characters by using their alias. For example, in Figure 3 (2), the order is `caption I | L` indicating that each sentence will contain four elements: first a series of textual characters of type caption, then a series of icons of type I, then a separator | and finally a series of icons of type L. IsoTex interprets each sentence content (Figure 2) to display it with the appropriate characters, computing the number of icons by the corresponding weight of the character.

The parameter `[align]` (Figure 5 left) specifies the alignment of each visual component in a sentence. In our example, the main sentence is divided into four sections, I is aligned to the right and L is aligned to the left, resulting in Figure 2. Borrowing from the LaTeX notation, we use the character | in the parameter alignment to indicate it applies to all sentences in the paragraph.

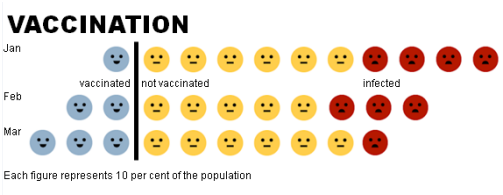
Fig. 5. Example of `[align]` and `[width]`

Paragraphs contain sentences and enable the creator to specify a height and width and thus determine how sentences are wrapped to fit within the paragraph boundaries.

The parameter `[width]` (Figure 5 right) is important when displaying large numbers of icons. In addition to absolute pixel number, it also enables wrapping at a sentence or character level. Thus, `[width=1sentence]` indicates that each sentence starts on a newline while cmd[width=10characters] indicates that a maximum of 10 characters will be displayed before going to the next line.

3.4 Examples

Figure 6 illustrates an iterated design from Figure 2. Specifying different sets of characters, sentences patterns and alignments, as well as paragraphs structures enable to a wide range of designs, a subset can be found at the end of this paper (Figure 9) and in the accompanying website http://isotex.github.io.




```
% Copy a character to add a category
\character[alias=new]{person.png}
% Add category to sentence structure and alignment
\sentence[characters= I | L new, align= right | left left]{}
% Drag and drop new icons
% Update data, title and labels
\sentence{Jan, 10, 60, 30}
...
```

Fig. 6. Iterating from Figure 2 with edits in bold

4 ISOTEX EDITOR

Coupled with the language, IsoTex offers an interactive editor to manipulate directly the visual output.

Coordination. The editor provides a brushing and linking between the textual markup language and visual isotype output view. It features two levels of coordination: highlighting content, and formatting. For example, brushing a row of icons with mouse over highlights the corresponding data content in a sentence, while selecting it with mouse click highlights the corresponding formatting (font and character).

Direct manipulation. The editor affords output-directed programming via direct manipulation of visual elements; changes are directly reflected in the textual specification. Dragging an image file on an icon updates it. The creator may also select from suggested icons and colors available in the interface. Note that we extracted these palettes from Otto Neurath's considerable corpus [7, 18] and propose over 1200 icons. Additional interactions are possible on the bounding boxes of elements to resize them (by dragging a corner) or to add padding (by dragging sides).

Note the textual specifications affords a trivial mechanism to save a complete history of changes and provide undo and redo for both interactions and textual edits.

Bidirectional workflows. Bidirectionality helps take advantage of both types of interactions: direct interaction from the editor that does not require knowledge of the correct keywords or following an exact syntax, coupled with the higher precision (e.g. typing an exact value) and reuse available by editing and copying portions of textual specifications. As a walk-through example, we describe how Emma, an illustrator with no knowledge of programming, can take advantage of working with both views (Figure 7).

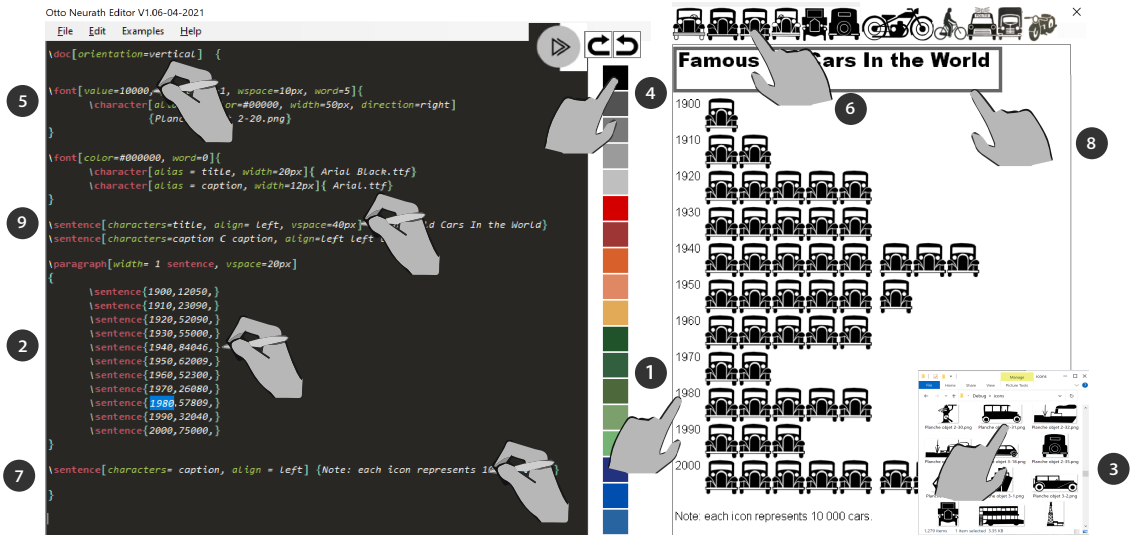


Fig. 7. Bidirectional workflow consisting of direct manipulation of visual elements and edition of the textual specification.

Starting from a code sample for a simple isotype (e.g. Figuer 9 middle), Emma learns how visual and textual elements correspond to each other using the brushing and linking mechanism (1). Emma then types in her new data (2). She uses copy&paste to add sentences, as her data has more years than the sample she started with. The corresponding isotype is updated featuring more rows, confirming that she is at the right place in the language.

Emma switches to edit the visuals. She finds a suitable icon from her hard disk and uses drag&drop to add it directly on the isotype (3). The isotype updates to feature this icon. Emma then experiments with multiple colors by using drag&drop, adding a color patch from the provided palette directly to one of the icons (4).

Emma now wants to fix her visualization because it contains too many icons, making it difficult to estimate and compare years. She finds the weight parameter and types multiple values, looking at the resulting output, until she finds the best compromise (5).

Selecting an icon in the editor provides a suggested icon palette, Emma experiments with a few different ones by dragging and dropping (6). She settles on one with a square aspect ratio as it make her isotype more compact.

Emma returns to the textual editor to add the legend to decode the isotype (7). Looking at the results, she realized that the title needs more padding. Emma selects it in the editor and drags the bottom of its bounding box to add more padding (8). A new parameter is automatically added to the specification. Emma returns to the language to try different values (9) finding the perfect amount of spacing.

5 IMPLEMENTATION

We implemented a functional prototype of IsoTex in C#, which allowed us to gather feedback from our target users and iterate on the design of the language and editor. During our design process, we strove to balance complexity, verbosity, and expressiveness of the language. The *complexity* of the language comes from nesting elements and integrating many parameters, such as controlling the reading order of displayed characters in the sentences (e.g. from right to left, bottom to top). The *verbosity* is greatly reduced when adding default values to parameters in the implementation, such as assigning a default color or spacing. Creating a variety of examples led us to reasonable defaults.

5.1 IsoTex Parser

Our language has a mark-up based structure (inspired by LaTeX), where the special character '\' introduce each keyword followed by options encapsulated by '[]' and content bounded by '{}'. In order to turn our textual description into computed compatible structures, we implemented a sequential parser. We opted for a double pass parsing process: first, each keyword, its parameters and content are stored in a data structure; and second, each data structure is then turned into a visual representation. This double pass is necessary as components are nested and thus depend on each other: paragraphs contain sentences which contain characters. Our parsing process operates as follows:

(1) As an initial step, we first parse the *characters* (i.e. fonts of image).
(2) We parse every *font keyword* and propagate their options into the nested character items. The language property aggregates options to make it more compact.
(3) We parse the remaining *character items* which are not nested into fonts.
(4) We parse *sentences* and store them. Sentences which only have options are called *prototype*. Prototypes will propagate their option to the other sentences unless the considered option already exists.
(5) We parse the *paragraphs* which contain sentences.

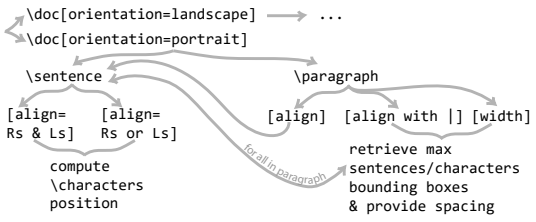
During this sequential parsing method, all visual items (character, paragraph and sentence) are stored an ordered list, the visual item list, which will be processed in the compilation step.

5.2 Document Compilation and Rendering Layout

Once the parsing process is completed, our system will compile every item of the visual item list. Each font is adjusted in size and color when specified in the optional parameters. We compute the layout using the following process: (1) computing individual bounding boxes for each set of characters, (2) identifying the paragraph layout from the set of boxes, and (3) assigning individual elements to their positions. Note that the width of paragraphs can be specified in sentences or characters, which individually vary in size (e.g. visual and textual characters have different sizes depending on content). To provide more regular layouts, we compute an average size for the number of characters provided in pixels, then we can slightly adjust the number of characters on each line to remain close to this average size (e.g. last example in Figure 9).

Computing the layout is the core operation for producing isotypes. This process is facilitated by our parsing and compilation steps. At this stage, we know the bounding box dimensions of each visual item (i.e. the available space filling) and its internal layout (i.e. alignment).

For the final step of the visual rendering of the isotype we use a state machine for dealing with the combination of layout options (e.g., alignment, separators, width). The figure on the right provides a high-level description of state machine. Note that we removed many parameters at the sentence level such as character reading order and word spacing to provide a readable outline of the process.



5.3 Interactive features

To provide a direct binding between every visual structure (icons, text) and their textual sibling in the parsed document, the implementation maintains a double hierarchical structure. Editing the specification alters the textual element hierarchy, which then triggers changes in the visual element hierarchy, causing an update of the visual preview. Respectively, interacting with a visual element alters the visual element hierarchy, which then triggers the textual element hierarchy causing the text specification to be updated.

To enable users to recolor icons, we follow a simple pixel based image processing where we extract luminosity, saturation, and hue of each pixel. If the considered pixel is black, we replace it with the novel color. If not, we maintain its saturation and luminosity but replace its hue. This simple algorithm provides reasonable results for most cases.

To suggest alternative icons, we compute a distance factor from a selected icon to our corpus. This distance is a pixel based comparison between normalized images. We first scale every image so that it shares the same width while keeping its aspect ratio. We vertically center every image with white space filling. We then sum up the color difference of every pixels. Such simple distance computation works well as a first approximation for finding visually similar icons, but can be improved by adding the semantic of the image.

The system also supports undo/redo by storing our language content at every stage of its textual or visual evolution. The user can also navigate within a temporal history (i.e., previous versions) of the language edits.

6 QUALITATIVE FEEDBACK

To gather feedback on the IsoTex language and editor, we recruited a convenience sample of five participants from diverse backgrounds: a designer (D), a communication professional (C), a data scientist (S), a visualization practitioner (V), and a web developer (W). We conducted interviews lasting about 1 hour via remote video call.

In the first half of the session, the experimenter told participants to imagine a colleague coming to them to create the isotype of Figure 2 and asked them how they would recreate it given their current workflow. The experimenter then showed three iterations of the isotype (Figure 8). For each, the experimenter asked participants to describe the tools and type of work needed to adjust their initial design. In the second half of the session, the experimenter described the language and interactions before asking participants targeted questions and changes to the specification.

Fig. 8. Study material: reference isotype and design alternatives (1) icon and color, (2) data unit, and (3) adding more data

6.1 Current Workflows and Pain Points

Every participant described *different tools and different workflows* to create an isotype. For example, the designer said that she would only use Adobe Illustrator, making all data computations of units on the fly (on paper), while the communication professional explained a 2-step process: creating bar charts in MS Excel to figure out the shape of the data, before importing them into MS Powerpoint to lay icons above each bar. Other participants mentioned ggplot, MS Paint, and D3/Vega and HTML/CSS+Javascript for participants with programming knowledge.

Independent of their workflows and tools, all five participants started the session by stating that it would be relatively easy for them to reproduce the isotype. However, as the experimenter asked them to alter their designs, they all expressed the need for substantial effort and time:

Iterating on icons is tedious. Two participants relying on more generic tools for laying out icons (such as MS PowerPoint) noted an extremely tedious process, requiring basically starting the design from scratch. Note that participants with programming knowledge or graphics editing tool knowledge found iterating over visuals relatively trivial, apart from fragmenting icons.

Iterating on data units is error prone. For participants not well versed in programming, iterating on the unit of data encoded by icons felt the more error prone part of the process. D described a high mental load, while S and C mentioned the risk of back and forth between data and visual tools leading to possible errors. Participants with programming knowledge (V, W) noted that it was likely they would need to write specific code at this point.

Adding data is a deal-breaker. Non-programmers all commented that adding new data to the isotype was too much effort. D commented *"at this point, if you want to do things like that, I would seek to partner with a developer"*. Note that programmers also described the need to start more substantial development: *"adding another series would probably introduce a whole new concept that I need to build. This is when you start developing your own framework"*.

6.2 Feedback on IsoTex

Document terminology is simple to understand. All participants commented positively on the parallels between textual document and isotype. A single participant (V) had prior knowledge of LaTeXand found the parallel *"brilliant"*. Non-programmers understood the notion of characters representing icons and could decipher sentence patterns and alignments. While they did not feel like they could write in this language from scratch, they felt comfortable editing existing specifications. They all mentioned that our highlighting function between language and corresponding visual elements was extremely helpful.

A markup language feels like code. Our non-programmers all mentioned that IsoTex felt like code. D and C mentioned it reminded them of wordpress and the ability to switch from a rich text format to underlying HTML. However, C noted that the simple keywords used made it *"far less daunting than code"*. D commented that she would definitely want to use it to craft isotype for a project she was working on and requested the application.

Bidirectionality is critical. D commented that she did not want to *"live in the code"* as that would *"disconnect [her] from the visuals"*. The direct manipulation afforded in the editor was really important to preserve such immersion in the visuals. The other non-programmers explained that the language specification was succinct and *"readable once you get the hang of it"* (C). They also noted that many parameters were hidden. Making these visible after interacting with visuals felt like a good solution to preserve readability of the specification while providing fine control when needed.

Non-programmers could work with IsoTex after 15 minutes. In the last part of the session, the experimenter asked participants how they would alter the specification for different iterations (Figure 8). All participants could successfully identify how to make these changes.

7 DISCUSSION AND FUTURE WORK

Isotypes appear deceptively simple to create and people use many different types of tools to create them (generic productivity tools, visualizations tools, graphics editing tools, or programming libraries), depending on what they are most familiar with. However, exploring isotype variations requires substantial effort with any of these tools, whether it be because of repetitive interactions, or difficulty in writing specific lines of code.

These efforts are likely to discourage many from exploring design alternatives. Yet, **exploring this design space is critical** to create high quality isotypes, which must strike a balance between many aesthetic, perceptual and functional criteria: a meaningful and compatible (aspect ratio) palette of icons to encode data quantities, an understandable data unit enabling rapid estimations and comparison, the right number of icons to enable rapid counting yet with enough precision to accurately convey the data, as well as an adequate layout to support comparing quantities and perceiving higher level trends.

Overall, the feedback we gathered suggests that IsoTex enables people to rapidly explore the design space, independently of their professional and technical background. These initial comments suggest that people can **transfer their knowledge** from the well familiar document terminology to the description of isotypes. Coupled with a visual editor enabling the modification of the specification, it appeared an appealing tool for people with little to no knowledge of programming.

Our design iterations among authors, as well as the informal feedback from users with diverse backgrounds, made us realize that **succinctness is not always preferable**. For example, nesting paragraphs to create more complex structures is extremely powerful, but resulting structures, while succinct, are difficult to grasp for non-programmers. Alternatives closer to textual document layout (such as enabling multiple columns by specifying a document width of 2 or 3 paragraphs) are likely preferable.

In the future, we aim at extending IsoTex to integrate automated guidance (e.g., on the choice of compatible icons and appropriate units) as well as provide feedback on the readability of the resulting visual (e.g., gap between the raw data and the perceived quantity). We will also explore if IsoTex is extensible to other families of visualizations.

8 CONCLUSION

Isotypes are omnipresent visuals for data storytelling. While these representations can be created from a large range of tools, none of them affords flexible and rapid iteration of its design. To close this gap, we proposed IsoTex, a declarative markup language describing isotypes as documents, coupled with a live visual editor affording direct manipulation of content. Preliminary feedback suggest that IsoTex is a promising tool for a large audience.

REFERENCES

[1] Basak Alper, Nathalie Henry Riche, Fanny Chevalier, Jeremy Boy, and Metin Sezgin. 2017. Visualization Literacy at Elementary School. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 5485–5497. https://doi.org/10.1145/3025453.3025877

[2] Fereshteh Amini, Nathalie Henry Riche, Bongshin Lee, Christophe Hurter, and Pourang Irani. 2015. Understanding Data Videos: Looking at Narrative Visualization through the Cinematography Lens. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 1459–1468. https://doi.org/10.1145/2702123.2702431

[3] Alex Bigelow, Steven Drucker, Danyel Fisher, and Miriah Meyer. 2014. Reflections on How Designers Design with Data. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces* (Como, Italy) *(AVI '14)*. Association for Computing Machinery, New York, NY, USA, 17–24. https://doi.org/10.1145/2598153.2598175

[4] Michelle A Borkin, Zoya Bylinskii, Nam Wook Kim, Constance May Bainbridge, Chelsea S Yeh, Daniel Borkin, Hanspeter Pfister, and Aude Oliva. 2015. Beyond memorability: Visualization recognition and recall. *IEEE transactions on visualization and computer graphics* 22, 1 (2015), 519–528.

[5] Michelle A Borkin, Azalea A Vo, Zoya Bylinskii, Phillip Isola, Shashank Sunkavalli, Aude Oliva, and Hanspeter Pfister. 2013. What makes a visualization memorable? *IEEE transactions on visualization and computer graphics* 19, 12 (2013), 2306–2315.

[6] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D³ Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec. 2011), 2301–2309. https://doi.org/10.1109/TVCG.2011.185

[7] Christopher Burke, Eric Kindel, and Sue Walker (Eds.). 2013. *Isotype: design and contexts 1925 1971*. Hyphen Press, London. http://centaur.reading.ac.uk/34937/

[8] Matt Conlen and Jeffrey Heer. 2018. Idyll: A Markup Language for Authoring and Publishing Interactive Articles on the Web. In *ACM User Interface Software & Technology (UIST)*. http://idl.cs.washington.edu/papers/idyll

[9] Pierre Dragicevic, Stéphane Huot, and Fanny Chevalier. 2011. Gliimpse: Animating from Markup Code to Rendered Documents and Vice Versa. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. Association for Computing Machinery, New York, NY, USA, 257–262. https://doi.org/10.1145/2047196.2047229

[10] Steve Haroz, Robert Kosara, and Steven L Franconeri. 2015. Isotype visualization: Working memory, performance, and engagement with pictographs. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 1191–1200.

[11] Jeffrey Heer and Michael Bostock. 2010. Declarative Language Design for Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (Nov. 2010), 1149–1156. https://doi.org/10.1109/TVCG.2010.144

[12] Brian Hempel, Justin Lubin, and Ravi Chugh. 2019. Sketch-n-Sketch: Output-Directed Programming for SVG. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 281–292. https://doi.org/10.1145/3332165.3347925

[13] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[14] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. 2020. mage: Fluid Moves Between Code and Graphical Work in Computational Notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 140–151.

[15] Nam Wook Kim, Eston Schweickart, Zhicheng Liu, Mira Dontcheva, Wilmot Li, Jovan Popovic, and Hanspeter Pfister. 2016. Data-driven guides: Supporting expressive design for information graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 491–500.

[16] Huiyang Li and Nadine Moacdieh. 2014. Is "chart junk" useful? An extended examination of visual embellishment. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 58. SAGE Publications Sage CA: Los Angeles, CA, 1516–1520.

[17] Zhicheng Liu, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. 2018. Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173697

[18] Vossoughian Nader and D'Laine Camp (Eds.). 2008. *Otto Neurath: the language of the global polis*. Rotterdam: NAi Publishers, Rotterdam.

[19] Nathalie Henry Riche, Christophe Hurter, Nicholas Diakopoulos, and Sheelagh Carpendale. 2018. *Data-Driven Storytelling* (1st ed.). A. K. Peters, Ltd., USA.

[20] Hugo Romat, Nathalie Henry Riche, Christophe Hurter, Steven Drucker, Fereshteh Amini, and Ken Hinckley. 2020. Dear Pictograph: Investigating the Role of Personalization and Immersion for Consuming and Enjoying Visualizations. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376348

[21] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2017). http://idl.cs.washington.edu/papers/vega-lite

[22] Jacob VanderPlas, Brian E Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. 2018. Altair: interactive statistical visualizations for Python. *Journal of open source software* 3, 32 (2018), 1057.

[23] Bret Victor. [n.d.]. Tangle. http://worrydream.com/Tangle/.

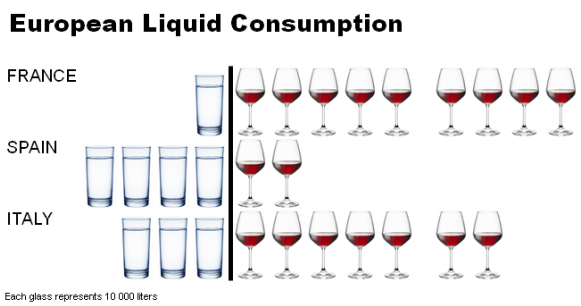
[24] Yun Wang, Haidong Zhang, He Huang, Xi Chen, Qiufeng Yin, Zhitao Hou, Dongmei Zhang, Qiong Luo, and Huamin Qu. 2018. InfoNice: Easy Creation of Information Graphics. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173909

[25] Hadley Wickham. 2006. An introduction to ggplot: An implementation of the grammar of graphics in R. *Statistics* (2006).

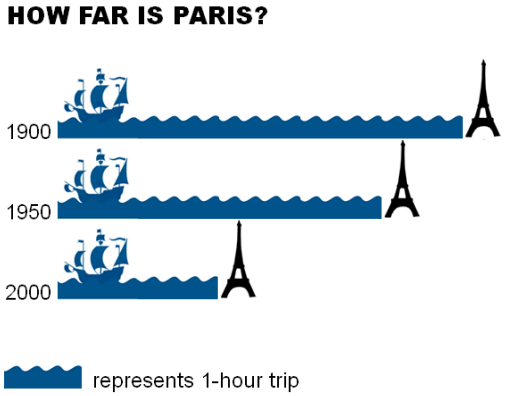
[26] Haijun Xia, Nathalie Henry Riche, Fanny Chevalier, Bruno De Araujo, and Daniel Wigdor. 2018. DataInk: Direct and Creative Data-Oriented Drawing. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173797

[27] Jiayi Eris Zhang, Nicole Sultanum, Anastasia Bezerianos, and Fanny Chevalier. 2020. DataQuilt: Extracting Visual Elements from Images to Craft Pictorial Visualizations. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376172

```
\doc[orientation=portrait]  {

\font[weight=10000, word = 5,  wspace=20px, width=80px]{
       \character[alias=Water] {water.png}
       \character[alias=Wine]  {wine.png}}

\font[color=#000000, word=0]{
       \character[alias = title, width=20px]{ Arial Black.ttf}
       \character[alias = country, width=14px]{ Arial.ttf}
       \character[alias = caption, width=8px]{ Arial.ttf}}

\character[alias = |, width = 5px, weight=1, clip=1, color=#000000, height=max]{|}

\sentence[characters=country Water | Wine, align= topleft right | left] {}

\sentence[characters= title, align = topleft, interline = 30px]
{European Liquid Consumption}

\paragraph[width= 1 sentence, vspace=10px]{
       \sentence{FRANCE,12030, 1, 95030}
       \sentence{SPAIN,40045, 1, 24200}
       \sentence{ITALY, 32021, 1, 72056}
}
\sentence[characters= caption, align = left]
{Each glass represents 10 000 liters}
}
/
```




```
\doc[orientation=portrait]  {

\font[wspace=0px, width=90px]{
       \character[alias=B, weight=1] {boat.png}
       \character[alias=S, weight=60] {sea.png}
       \character[alias=P, weight=1] {france.png}}

\font[color=#000000, word=0]{
       \character[alias = title, width=20px]{ Arial Black.ttf}
       \character[alias = caption, width=18px]{ Arial.ttf}}

\sentence[characters=caption B S P, align= bottomLeft left left left] {}

\sentence[characters= title, align = topleft, interline = 30px]
{HOW FAR IS PARIS?}

\paragraph[width= 1 sentence, vspace=10px] {
       \sentence{1900,1,240,1}
       \sentence{1950,1,180,1}
       \sentence{2000,1,60,1}
}

\sentence[characters=S caption, align=Left bottomLeft]
{60, represents 1-hour trip}
}
```




```
\doc[orientation=portrait]  {

\character[alias=icon, weight=1, color=#333333, width=35px] {p.png}

\font[color=#333333]{
  \character[alias = title, width=22px]{ Arial Black.ttf}
  \character[alias = text, width=20px]{ Arial.ttf}
}


\sentence[characters= title, interline = 30px]
{A YEAR OF POLICE FATALITIES ~ 2015}


\paragraph[width= 55 character]{
      \sentence[characters=text]{DEC}\sentence[characters=icon]{27}
      \sentence[characters=text]{NOV}\sentence[characters=icon]{70}
      \sentence[characters=text]{OCT}\sentence[characters=icon]{27}
      \sentence[characters=text]{SEP}\sentence[characters=icon]{81}
      \sentence[characters=text]{AUG}\sentence[characters=icon]{94}
      \sentence[characters=text]{JUL}\sentence[characters=icon]{103}
      \sentence[characters=text]{JUN}\sentence[characters=icon]{65}
      \sentence[characters=text]{MAY}\sentence[characters=icon]{71}
      \sentence[characters=text]{APR}\sentence[characters=icon]{84}
      \sentence[characters=text]{MAR}\sentence[characters=icon]{92}
}
}
```

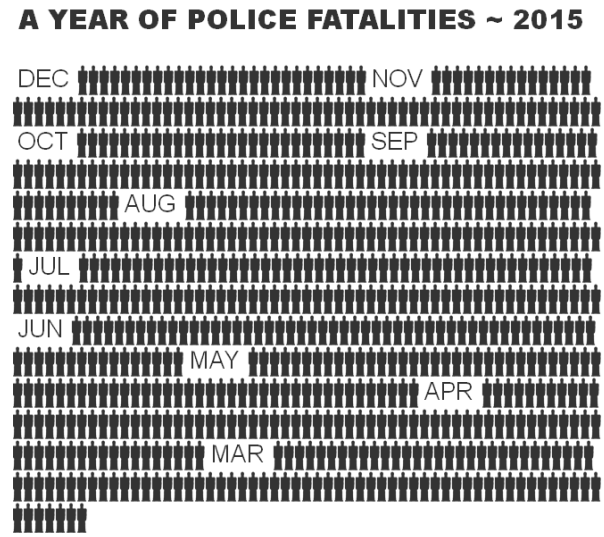


Fig. 9. Appendix. Examples of Isotypes generated with IsoTex