Dynamic 2.5D Treemaps using Declarative 3D on the Web

Daniel Limberger*
Hasso Plattner Institute
University of Potsdam

Willy Scheibel
Hasso Plattner Institute
University of Potsdam

Stefan Lemme
DFKI
Saarland University

Jürgen Döllner
Hasso Plattner Institute
University of Potsdam

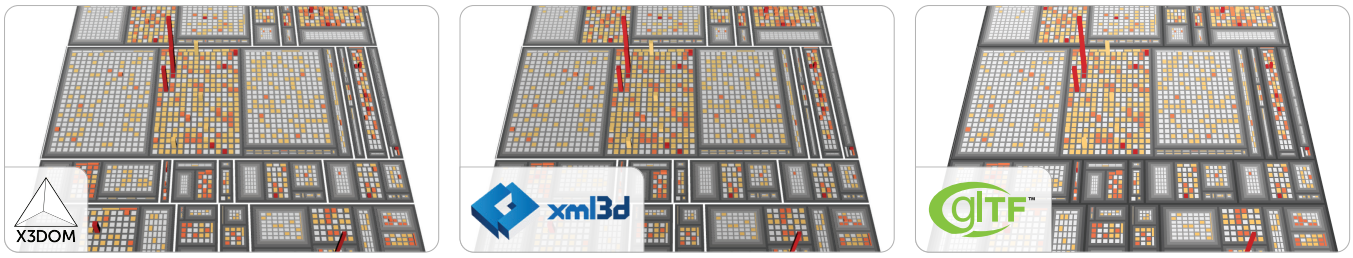

Figure 1: *Interactive 2.5D treemap clients (with dynamic attribute mapping) based on X3DOM, XML3D, and glTF developed in our study.*

Abstract

The 2.5D treemap represents a general purpose visualization technique to map multi-variate hierarchical data in a scalable, interactive, and consistent way used in a number of application fields. In this paper, we explore the capabilities of Declarative 3D for the web-based implementation of 2.5D treemap clients. Particularly, we investigate how *X3DOM* and *XML3D* can be used to implement clients with equivalent features that interactively display 2.5D treemaps with dynamic mapping of attributes. We also show a first step towards a *glTF*-based implementation. These approaches are benchmarked focusing on their interaction capabilities with respect to rendering and speed of dynamic data mapping. We discuss the results for our representative example of a complex 3D interactive visualization technique and summerize recommendations for improvements towards operational web clients.

Keywords: 2.5D Treemap; Dec3D; X3DOM; XML3D; glTF

Concepts: •**Human-centered computing** → **Visual analytics;** *Web-based interaction;*

1 Introduction

Treemaps [Shneiderman 1992] provide effective means to display, explore, and analyze multi-variate large hierarchical data, e.g., large-scale software system data [Langelier et al. 2005; Bohnet and Döllner 2011] and business data [Vliegen et al. 2006]. Treemaps use space-restricted, recursively nested sets of rectangles, which express parent-child relationships among nodes; the rectangles' sizes are proportional to per-node weights. Data associated with nodes, denoted by *attributes*, can be mapped by means of visual variables [Carpendale 2003] of treemaps, e.g., rectangle size, color, texture, and shading. *2.5D treemaps* extend treemaps by using the third dimension: rectangles are extruded to 3D blocks, keeping the regular treemaps two-dimensional reference space and layout [Bladh et al. 2004]. Blocks emphasize the hierarchical nesting of inner nodes and introduce another strong visual variable, namely *height*, for the leafs. Thus, 2.5D treemaps allow us to independently map three attributes (size, color, height) and to visually relate these attributes in a single view. Technically, 2.5D treemaps are represented by 3D scenes and often ambiguously denoted by "3D treemaps".

A growing number of applications and systems can take advantage of 2.5D treemaps providing means to interactively explore and analyze large-scale hierarchical data. Interactivity and scalability are key to the user to gain insights, to uncover structures and patterns as well as to acquire knowledge by reasoning. Insofar, 2.5D treemaps are representative for a large number of dedicated interactive data visualization techniques. Incorporating a 2.5D treemap as a visualization component in web-based applications is faced by a number of challenges from a software engineering point of view. In particular, if implementations are based on imperative 3D graphics programming, typical problems arise related to code complexity, interoperability across platforms, integration into web-based workflows, and software maintenance. *Declarative 3D* provides an alternative approach (Figure 1) investigated in this paper. We present and evaluate implementations of 2.5D treemap web clients based on three different technologies X3DOM, XML3D, and glTF. We also discuss how to efficiently and dynamically map attribute data to visual variables. Our evaluation focuses on three main aspects:

- *Rendering Performance*: User interaction requires interactive frames rates and short data transmission times, taking into account that treemaps visualize up to $10^4 - 10^7$ items.
- *Dynamic Data Mapping*: The clients need to support dynamic attribute mapping on a per-item basis, e.g., adding additional data, selecting and filtering individual items.
- *Declarative Paradigm*: The visualization clients should be fully based on declarative 3D graphics.

2 Related Work

The treemap [Shneiderman 1992] represents a well-known technique to depict hierarchically structured information in a space-restricted, space-filling way using recursive inclusion. Various layout algorithms have been investigated with different characteristics, e.g., aspect ratio of resulting rectangles, layout stability and robustness [Tak and Cockburn 2013]. Apart from a large number of stand-alone implementations, a first scalable web-based 2.5D treemap [Limberger et al. 2013], used to visualize data about software systems and development processes, efficiently encodes data transferred to clients and uses vertex attribute arrays for rendering.

While WebGL enables most browsers to directly render 3D gra-

*e-mail:daniel.limberger@hpi.de

Web3D '16,, July 22-24, 2016, Anaheim , CA, USA
ISBN: 978-1-4503-4428-9/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2945292.2945313

phics inside the HTML <canvas> element, it is based on a low-level imperative API. The use of such APIs, however, can be considered to be orthogonal to most existing web technologies, which include contents in web documents and get modified via DOM API. The "Declarative 3D for the Web Architecture W3C Community Group"provides platforms to embed 3D graphics within the HTML DOM [Jankowski et al. 2013]: X3DOM [Behr et al. 2009] and XML3D [Sons et al. 2010]. X3DOM implements a subset of the nodes specified by X3D, an existing ISO standard [Web3D Consortium 2008], for the web. In contrast to X3DOM, which brings an existing format into the web, XML3D provides a minimal set of elements as an extension to HTML5. Thereby, XML3D seamlessly integrates with the web technology stack and leverages HTML, CSS, and JavaScript events. The Khronos Group released the OpenGL Transmission Format *glTF* [Khronos Group 2015], which will allow applications to receive and process standard format 3D assets and, thereby, to reuse the ecosystem of asset pipeline tools. glTF supports extensions, e.g., extern and binary assets. Compared to X3DOM and XML3D, glTF is only a transmission format, not intended to be directly renderable, as it has to be converted into renderer-specific scene descriptions. As a consequence, it is difficult to directly modify the 3D scene at a high level of abstraction.

Since browser vendors stated that native support of declarative 3D is not their priority [Jankowski et al. 2013], supplementary support by means of WebGL-based *Polyfill* implementations [Sons et al. 2013] are provided. In terms of glTF, we take advantage of a renderer based on three.js[1].

3 2.5D Treemaps using X3DOM

X3D documents typically contain a <scene>, a <viewpoint>, and various <transform>s and <shape>s. We have applied three different approaches for encoding the geometry of 2.5D treemaps: (1) multiple <box> tags, (2) a single unit cube specified by a single <indexedfaceset> that is reused, and (3) a pre-baked buffer via a single <indexedfaceset> tag. Attribute mapping to the visual variables height, color, and ground size is explicitly resolved and, thus, directly specified in X3D properties. In practice, the attribute mapping to the cuboids' ground size is often fixed to a single attribute, i.e., it does not change for various map themes. X3DOM provides a special "turntable" navigation that allows users to control azimuth and altitude. The latter can be further constrained by lower and upper angles to restrict the camera position to the upper hemisphere, which is preferred for 2.5D treemaps. For picking of individual data elements, e.g., to provide selection and filtering mechanics as well as additional node information, we leverage DOM event handlers and process the desired element's attributes appropriately. This design allows us to declaratively specify a 2.5D treemap depicting several thousand data elements via X3DOM.

Boxes. This approach uses the <box> tag as base geometry for each node of the treemap. The base tag of a treemap node is a <transform> for the position and size transform of the shared base geometry. <appearance> and <material> tags define a node's color, specifically the diffuseColor attribute is used. A <shape> with the <box> as child reuses the base geometry. Updating treemap nodes is restricted to changes to <transform> attributes for height and <material> attributes for the color. Similarly, we use emissiveColor of <material> for highlighting.

Shared Geometry. A 2.5D treemap is mainly composed of boxes and typically viewed from the top hemisphere. Hence, the bottom faces can be omitted to optimize the geometric representation. The <indexedfaceset> tag is used to define five faces for each node. The normals are defined for each face; setting its normalPerVertex attribute to false. This shape is then reused for all treemap nodes using the def and use attributes. The usage in the X3DOM hierarchy and the updates are similar to the boxes approach.

Pre-Baked Buffer. Similar to geometry encoding of WebGL-based rendering systems for 2.5D treemaps [Limberger et al. 2013], XML3D enables geometry specification by means of coordinate, normal, color, and index buffers via the <indexedfaceset>. The five individual faces of every block (omitting bottom faces) are encoded via 3D points in a buffer represented by the points attribute of a single <coordinate> tag. The points are then referenced by their positions within the buffer using the face sets coordindex attribute. For encoding of colors and normals there are two ways: (1) reusing the coordinate indices by providing a color value and a normal for each vertex individually or (2) define the available colors and normals once and provide additional index buffers for coordinate-based reference. Either way, colors are encoded in the color attribute of a single <colorrgba> tag, normals in the vector attribute of a single <normal> tag. The colorIndex and normalIndex attributes can then be used for optimized indexation. With the respective knowledge of each buffers structure, we implement dynamic remapping by changing the values at appropriate positions via JavaScript. Even though the identification of single treemap nodes is possible, X3DOMs picking buffer implementation cannot be used to identify nodes by picking: the suggested encoding of identifiers via color is not an option since color itself is required as visual variable. Using the idBuf does not help either since only world space positions are returned and a position based retrieval of individual nodes is nontrivial.

4 2.5D Treemaps using XML3D

With respect to data handling XML3D is flexible due to its integration of Xflow [Klein et al. 2013]. However, every renderable object in an XML3D scene is represented using a <mesh> element to achieve individual interaction (i.e., event handler). Each mesh is fed with geometry data, material properties, and a scene graph transformation. In the following, approaches using basic meshes, assets, and pre-baked buffers are discussed.

Basic Meshes Since 2.5D treemaps are exclusively composed of blocks a single base geometry per <mesh> element can be reused to minimize the memory footprint. This is done by referring to the same <data> element via document id in the src attribute. The material is shared by all blocks as well; it is attached once to the surrounding <group> element and derived by its children. For each mesh the material is configured by overriding the diffuseColor to customize its appearance. To highlight map elements either the material properties are changed or a different material is applied. For it, the elements material attribute is set, causing the material derived from the surrounding group to be overridden. Finally, a scene graph transformation is used to put each block in place and shape it accordingly. For that purpose, XML3D usually takes advantage of CSS 3D Transforms [Sons et al. 2013]. However, XML3D also offers the <transform> element for convenience and improved performance in interactive applications. Rather than frequently updating a mesh's style attribute, each mesh refers to its dedicated transform, which is updated in turn. As a consequence, it doubles the number of required DOM elements but also enables dataflow processing for the block transformations.

Assets With the introduction of configurable models by Klein et. al [2014], XML3D was extended by asset instancing. Hence, we

[1] http://threejs.org/examples/webgl_loader_gltf.html

Basis	Approach	DOMContent Loaded	First Frame Displayed	Continuous Rendering	Remapping all Colors	Remapping all Heights	Highlighting of one Leaf Node
X3DOM	**Boxes**	0.48s	4.05s	15fps	87ms	152ms	91ms
X3DOM	**Shared Geometry**	1.37s	3.60s	15fps	87ms	124ms	86ms
X3DOM	**Pre-baked Buffer**	0.63s	1.19s	60fps	112ms	552ms	696ms
XML3D	**Mesh**	1.59s	2.05s	30fps	173ms	98ms	37ms
XML3D	**Assets**	2.21s	2.70s	29fps	97ms	104ms	40ms
XML3D	**Pre-baked Buffer**	1.46s	2.78s	28fps	291ms	293ms	37ms
glTF	**Static**	0.24s	0.29s	60fps	–	–	–

Table 1: *All measurements were captured and averaged over* 1.000 *iterations for the same data set of 2990 nodes (358 parent and 2632 leaf nodes) with the same attribute mapping applied. All data was deployed on a server (http://hpicgs.github.io/web3d-treemaps/) and loaded/processed locally in Chrome (50.0.2661.87 64-bit) running on a Notebook (Intel Core-i7 6700HQ, 16GB RAM, Windows, Intel HD 530). For polyfill and DOM publicly available resources were used based on X3DOM 1.7.1, XML3D 5.1.4, and glTF 1.0 (with three.js r76).*

can wrap the block geometry into an `<assetmesh>` within a `<asset>` element. For each instance, we create a `<model>` element and configure it with child elements that are fed into the dataflow of the asset. Since the material is applied as before and derived from the surrounding group, we just set the `diffuseColor` as before. To put each block in place, we have exactly the same options as beforehand. Even though it is desirable to configure the transform via the asset interface as well, meshes and assetmeshes can refer to `<transform>` elements only by document ids, whereas their global scope prohibits this approach.

Pre-Baked Buffer In contrast to the previous approaches where the same block was transformed several times into its final place at runtime, in this approach we pre-compute the full geometry of the 2.5D treemap. For it, we duplicate the block and store its transformed geometry into a (large) buffer. We can take advantage of domain-specific dataflow operators to perform the block's transformation based on raw input data to be visualized. We could use a single `<mesh>` element for the treemap due to the baked transform, however this breaks individual interaction. Instead, we maintain one mesh per block, all referring to the same `<data>` element, but using indices to draw just the relevant subset of the buffer. These indices are in turn computed based on the id of the block to be represented. The material is applied as before and derived from the surrounding group. We can also just set the `diffuseColor` as before but this approach also allows for baking colors of the blocks into their vertices since the data is duplicated anyway.

5 2.5D Treemaps using glTF

To effectively encode 2.5D treemaps with glTF we rely on the static scene description functionality, i.e., no animations and skins have been used. The treemap and its rendering can be fully described using accessors, buffer views, buffers, materials, meshes, nodes, programs, scenes, shaders, and techniques. Although several different approaches to encode the treemap scene exist, the one with acceptable characteristics regarding loading time and rendering performance was chosen: a single mesh representing the whole 2.5D treemap. One disadvantage is the implementation of picking, for which we have to render an explicit id buffer and then perform texture lookups while picking. The resulting glTF description uses buffers containing the positions, normals, and additional attributes of a list of blocks. The used additional attributes are the color of the treemap node and its id, primarily used for picking. Buffer views and accessors for each buffer makes the vertices accessible to the rendering system. The scene contains one single mesh representing all treemap nodes as its primitives. It uses one material that provides the used technique and no additional values. The rendering of the whole scene uses one technique, which results in one vertex shader, one fragment shader and one program.

Due to the lack of a prevalent or official web-based glTF viewer, the effort to use glTF becomes comparable to a dedicated WebGL-based renderer and, therefore, we dropped it for evaluation in the light of declarative 3D.

6 Results and Discussion

For performance measurements, the clients processed and rendered the same attributed data set with 2,990 nodes. The results were obtained by manual instrumentation and suggest interactive frame rates on consumer hardware for all approaches (Table 1).

Similar to other available WebGL-based rendering libraries, XML3D, X3DOM, and glTF are designed for general 3D contents, which impedes use-case specific optimizations. While X3DOM features best loading performance, the visualization is ready in XML3D first (except for the pre-baked buffers). Here, XML3D probably benefits from its generic data handling that is close to actual WebGL buffers. While numerous approaches for handling complex 3D geometry exist, i.e., consisting of massive number of triangles or extensive texture data [Behr 2012; Limper et al. 2014; Sutter et al. 2014], in most cases they deal with comparatively few separate meshes. In data visualization the prerequisites are usually opposite: the geometry is simple but the number of meshes is large, which is not specifically addressed by any of these.

When it takes too long to load a web page, it is declared unresponsive by today's browsers, offering the user to cancel their rendering attempt. To bypass this kind of situation, we selected a data set that could be rendered with all declarative 3D approaches. For client-side computation of the treemap layout, all elements would have to be created dynamically, circumventing the page unresponsive behavior. However, this would affect the time to the first frame due to the additional required computations.

For both declarative 3D approaches, the deployment was simple. X3DOM has a slight advantage for 2.5D treemaps (in terms of expressiveness) because it includes the `<box>` element, not requiring an explicit vertex specification. Common functionality, such as picking, further supports this. In addition, Xflow's data processing capabilities in XML3D provides an appreciated flexibility for processing visualization data, e.g., it can overcome the lack of geometry shaders in WebGL.

Changing the geometry in the scene can be easily accomplished in both X3DOM and XML3D by manipulating the DOM. While most

of our approaches require custom JavaScript code to apply these manipulations throughout several `<shape>` or `<mesh>` elements in the DOM, XML3D's flexible data composition system can be used to reduce the number of DOM manipulations.

Because declarative 3D implementations ease development, the time to create the the 2.5D treemap visualization clients was rather low (a few days). Using standard formats for such visualization makes them more accessible on the web, which was one of the goals for this case study. Declarative 2.5D treemap visualization could be easily integrated into existing web pages or published in a cloud-based collaboration platform. In addition, using standardized, open data formats simplifies archiving of visualization and there is no need for a proprietary, self-maintained viewer application.

7 Conclusions and Future Work

All presented approaches enable the embedding of 2.5D treemaps within the existing web technology stack (with a constraint of XML3D and X3DOM for interaction). This allows us to leverage existing web technologies such as JavaScript events for interaction handling. Features like built-in object picking, e.g., for selection and hovering of blocks, dynamic data remapping as well as basic navigation concepts components top off Declarative 3D to a full-fledged sandbox for rapid prototyping of visualization components. Hence, Declarative 3D provide a powerful alternative to dedicated renderers, realizing their limitations. Particularly, the large set of elements usually handled in data visualization points out a crucial shortcoming. Optimized approaches driven by use-cases will probably always outperform more generic approaches such as Declarative 3D. Nonetheless, these limits can be pushed much further to achieve sufficient performance. For instance, revisiting asset instances (i.e., optimize buffer usage and draw calls, geometry instancing) and a proper use of the respective shader stages (i.e., vertex displacement and color lookup tables) may offer another magnitude of performance. Once Xflow supports hassle-free parallel execution in mainstream browsers, this will become even more interesting. Last, enabling geometry instancing during rasterization would be good candidate for future work.

The open-source implementation is made available under MIT license and can be accessed via GitHub pages: https://hpicgs.github.io/web3d-treemaps/.

Acknowledgements

This work was partially funded by the Federal Ministry of Education and Research (BMBF), Germany, within the InnoProfile Transfer research group "4DnD-Vis" (www.4dndvis.de), and has received funding from the European Unions Seventh Framework Programme under grant agreement no. 632893 (FI-Core), and under grant agreement no. 641191 (CIMPLEX) in the European Unions H2020 Framework Programme.

References

BEHR, J., ESCHLER, P., JUNG, Y., AND ZÖLLNER, M. 2009. X3dom: A dom-based html5/x3d integration model. In *Proc. ACM Web3D*, 127–135.

BEHR, J., 2012. Declarative 3d workshop report. https://www.w3.org/community/declarative3d/2012/06/11/2012-declarative-3d-workshop-report/.

BLADH, T., CARR, D. A., AND SCHOLL, J. 2004. *Extending Tree-Maps to Three Dimensions: A Comparative Study*. 50–59.

BOHNET, J., AND DÖLLNER, J. 2011. Monitoring code quality and development activity by software maps. In *Proc. ACM MTD*, 9–16.

CARPENDALE, M. S. T. 2003. Considering visual variables as a basis for information visualisation. Tech. rep.

JANKOWSKI, J., RESSLER, S., SONS, K., JUNG, Y., BEHR, J., AND SLUSALLEK, P. 2013. Declarative integration of interactive 3d graphics into the world-wide web: Principles, current approaches, and research agenda. In *Proc. ACM Web3D*.

KHRONOS GROUP, 2015. Gl transmission format specification. https://github.com/KhronosGroup/glTF.

KLEIN, F., SONS, K., RUBINSTEIN, D., AND SLUSALLEK, P. 2013. Xml3d and xflow: Combining declarative 3d for the web with generic data flows. *IEEE Computer Graphics & Applications (CG&A) 33*, 5, 38–47.

KLEIN, F., SPIELDENNER, T., SONS, K., AND SLUSALLEK, P. 2014. Configurable instances of 3d models for declarative 3d in the web. In *Proc. ACM Web3D*, 71–79.

LANGELIER, G., SAHRAOUI, H., AND POULIN, P. 2005. Visualization-based analysis of quality for large-scale software systems. In *Proc. ACM ASE*, 214–223.

LIMBERGER, D., WASTY, B., TRÜMPER, J., AND DÖLLNER, J. 2013. Interactive software maps for web-based source code analysis. In *Proc. ACM Web3D*, 91–98.

LIMPER, M., THÖNER, M., BEHR, J., AND FELLNER, D. W. 2014. SRC - a streamable format for generalized web-based 3d data transmission. In *Proc. ACM Web3D*, Web3D '14, 35–43.

SHNEIDERMAN, B. 1992. Tree visualization with treemaps: A 2d space-filling approach. *ACM Trans. Graph. 11*, 1, 92–99.

SONS, K., KLEIN, F., RUBINSTEIN, D., BYELOZYOROV, S., AND SLUSALLEK, P. 2010. XML3D: Interactive 3d graphics for the web. In *Proc. ACM Web3D*, 175–184.

SONS, K., SCHLINKMANN, C., KLEIN, F., RUBINSTEIN, D., AND SLUSALLEK, P. 2013. xml3d.js: Architecture of a polyfill implementation of XML3D. In *Proc. IEEE SEARIS*, 17–24.

SUTTER, J., SONS, K., AND SLUSALLEK, P. 2014. Blast: A binary large structured transmission format for the web. In *Proc. ACM Web3D*, 45–52.

TAK, S., AND COCKBURN, A. 2013. Enhanced spatial stability with hilbert and moore treemaps. *IEEE Trans. Vis. Comput. Graph. 19*, 1, 141–148.

VLIEGEN, R., VAN WIJK, J. J., AND VAN DER LINDEN, E.-J. 2006. Visualizing business data with generalized treemaps. *IEEE Trans. Vis. Comput. Graph. 12*, 5, 789–796.

WEB3D CONSORTIUM, 2008. ISO/IEC 19775:200x, Extensible 3D (X3D). http://www.web3d.org/x3d/specifications/x3d_specification.html.