

A Modular Domain-Specific Language for Interactive 3D Visualization

Dominik Scholz
Visual Computing

TU Wien Informatics
Institute for Visual Computing and Human-Centered Technology
Research Unit of Computer Graphics
Supervisor: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Dipl.-Ing. Harald Steinlechner

Motivation

- enable 3D creation for non computer graphic experts
- ease sharing of interactive visualizations
- provide custom interaction techniques for domain experts
- extend visualizations by modular components

Case Studies

Case Study #1:

- simple example showing the system's general working
- wrtiting an interactive visualization with a custom module
- extend module with custom inputs and interactions

```

1 {
2   "camera": {"rotation": [-20, 0, 0], "position": [0, 10, 20]},
3   "sphere": {
4     "radius": 2.5,
5     "color": { "r": 0, "g": 1, "b": 0 },
6     "position": [0, 2.5, 0],
7     "onHover": {"color": [1, 0, 0]}
8   },
9   "events": {
10    "CLICK[m.sphere.radius < 10]": "{ sphere: { radius: m.sphere.radius + 1 }}",
11    "CLICK[p.shiftKey]": {
12      "sphere": {
13        "radius": 2.5,
14        "color": { "r": 0, "g": 1, "b": 0 }
15      }
16    },
17    "KEY_DOWN[p.key == '1']": "{ sphere: { color: { r: 1, g: 1, b: 0 }}}",
18    "KEY_DOWN[p.key == '2']": "{ sphere: { color: { r: 0, g: 1, b: 1 }}}",
19    "KEY_DOWN[p.key == '3']": "{ sphere: { color: { r: 1, g: 0, b: 1 }}}",
20    "KEY_DOWN[p.key == 'q']": "{ sphere: { color: { r: 1 - m.sphere.color.r }}}",
21    "KEY_DOWN[p.key == 'a']": "{ sphere: { color: { g: 1 - m.sphere.color.g }}}",
22    "KEY_DOWN[p.key == 'e']": "{ sphere: { color: { b: 1 - m.sphere.color.b }}}",
23  },
24  "pipeline": [
25    "scene",
26    "perspective-camera",
27    "sphere",
28    "render",
29    "raycaster",
30    "orbit-controls",
31    "key-listener",
32    "argumented-events",
33    "source"
34  ]
35 }

```

Configurable Data

Interactive Visualization

Pipeline Definition
(order of executed modules)

```
"KEY_DOWN[p.key == '1']": "{ sphere: { color: { r: 1, g: 1, b: 0 }}}",
```

Defining an interaction where the down press of the "1"-key corresponds to an update of the model to change its color to yellow. The beginning of the field name defines a message name, the argument in the square brackets specifies a dynamic comparison, and the value describes how the model gets updated when the message occurs and the argument evaluates to true.

```

1 {
2   "volume": {
3     "loadPath": "../data/stagbeetle832x832x494.dat",
4     "colorRange": [0, 1],
5     "renderStyle": "iso",
6     "threshold": 0.2,
7     "texture": "color"
8   },
9   "texturePaths": {
10    "color": "../data/cm_viridis.png",
11    "gray": "../data/cm_gray.png"
12  },
13  "camera": {
14    "position": [14, 54, -20],
15    "rotation": [-110, 13, 147],
16    "zoom": 4
17  },
18  "scene": {"grid": true, "backgroundColor": [0.1, 0.1, 0.1]},
19  "pipeline": [
20    "scene",
21    "volume-loader",
22    "texture",
23    "orthographic-camera",
24    "volume",
25    "render",
26    "orbit-controls",
27    "source"
28  ]
29 }

```

Defining textures as key-value pairs of the "texture-Paths"-field. The values of the pairs are paths to texture files, which get loaded and stored under the respective key names. These key names are variables and not pre-defined. The 'texture'-field can specify a texture as the transfer function by using the corresponding name.

```
"texture": "color"
```

```
"texturePaths": {
  "color": "../data/cm_viridis.png",
  "gray": "../data/cm_gray.png"
}
```

Case Study #2:

- shows praticality of our system for large data
- volume visualization
- change shader parameters through DSL
- DSL variables (fields that are not defined by a module)

Contribution

A declarative domain-specific language (DSL) for interactive 3D visualizations

- able to configure all stages of the visualization pipline
- human readable format (JSON) as host language
- users can create and configure interactive 3D visualizations

Modular system for computing visualizations using the DSL

- declarative design principles with functional data structures
- interfacing with imperative web environment

Visualization-creation environment

- allows quick prototyping of visualization design
- live DSL shows internal state and enables parameter changes

Case Study #3:

- dynamic encoding functions map data to a point cloud
- integration of Vega as a sub-DSL resulting in a linked view

```

1 {
2   "vega": {
3     "$schema": "https://vega.github.io/schema/vega-lite/v4.json",
4     "data": {"url": "../data/movies.json"},
5     "width": 400,
6     "height": 100,
7     "encoding": {
8       "x": {
9         "field": "value",
10        "title": "IMDB Rating",
11        "type": "quantitative"
12      },
13      "y": {"field": "density", "type": "quantitative"}
14    },
15    "transform": [{"density": "IMDB Rating", "bandwidth": 0.1}],
16    "layer": [
17      {
18        "selection": {
19          "brush": {"type": "interval", "encodings": ["x"]}
20        },
21        "mark": "area"
22      },
23      {
24        "transform": [{"filter": ("selection": "brush")}],
25        "mark": {"type": "area", "color": "goldenrod"}
26      }
27    ]
28  },
29  "pointcloud": {
30    "data": "m.vega.data.url",
31    "position": {
32      "x": "normalize('US_Gross')",
33      "y": "normalize('IMDB_Rating')",
34      "z": "normalize('Production_Budget')"
35    },
36    "color": {
37      "r": "brushed('IMDB_Rating') ? 1 : 0",
38      "g": "brushed('IMDB_Rating') ? 1 : 0.5",
39      "b": "normalize('IMDB_Rating')"
40    },
41    "size": 2,
42    "highlight": {
43      "color": {"r": 1, "g": 0, "b": 0},
44      "size": 4,
45      "index": 1704
46    },
47    "brush": [4.5, 6]
48  },
49  "camera": {
50    "position": [-156, 104, -35.000000000000036],
51    "rotation": [-120, -60, -132]
52  },
53  "pipeline": [
54    "scene",
55    "perspective-camera",
56    "pointcloud",
57    "render",
58    "vega",
59    "raycaster",
60    "orbit-controls",
61    "source"
62  ]
63 }

```

Linked View

2D Density Plot

3D Pointcloud

```
"color": {
  "r": "brushed('IMDB_Rating') ? 1 : 0",
  "g": "brushed('IMDB_Rating') ? 1 : 0.5",
  "b": "normalize('IMDB_Rating')"
},
```

Encoding functions define the mapping from the raw data to the visualization. The red and the green color channel of the pointcloud depend on the brushing range. The blue channel uses the "normalize" function, which maps the raw data values to a 0-1 range.

Approach

