

Appendix of ScaleTraversal

(Submission ID: 2963)

Compiler Design

We have built a light-weight compiler (Fig. 1), which is exploited to analyze the script codes and further mapped them to visual presentation codes and automatic virtual camera operations. The compiler consists of Syntax checking, camera tracking. The rendering component of ScaleTraversal is developed on GPU rendering and OpenGL libraries. For example, the rendering parts of the general-purpose data presentations are mainly implemented on OpenGL libraries. However, volume data rendering and the lighting effect are implemented based on GPU rendering, which makes the demonstration animation smooth.

The compiled results will be transferred to the rendering component to generate presentation animations. ScaleTraversal is designed like a programming library, which can be changed and added more rendering functions in future.

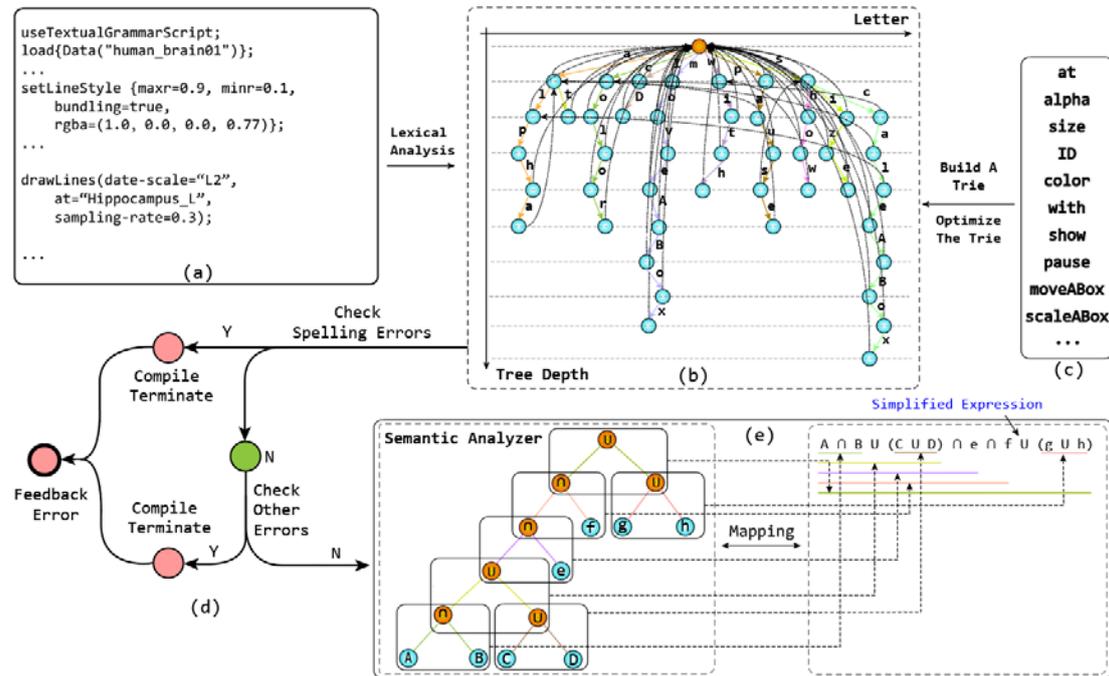


Figure 1. A design diagram of the compiler of ScaleTraversal. (a) A DSL script codes sample (b) A trie tree is constructed for lexical analysis. (c) Some extracted identifiers and keywords to be analyzed by the trie tree. (d) The state diagram of the spelling checking parser. (e) A binary tree is built to analyze the parallel expression semantically.

Super Object Combination Strategy

Typically, the CPU processes the data, state, and other information of one object each time to the command buffer as a “Draw Call”. “Draw Call” is a frequently-used data sharing strategies in GPU programming. They are appended to the command buffer by CPU, and the GPU reads them from the command buffer and executes them.

If there are a large number of individual objects in the rendering space, the CPU will spend a lot of time on pushing the “Draw Calls”, resulting in FPS decreasing across the overall animation. We create a super object combination strategy to save the data and state information of multiple objects in batch, while the CPU only needs to push the super object each time (G3). Several objects in an identical tissue in a given scale level can be combined into a super object, as shown in Figure 2. The strategy significantly reduces the number of “Draw Calls” and further increases the FPS of animation rendering (G3).

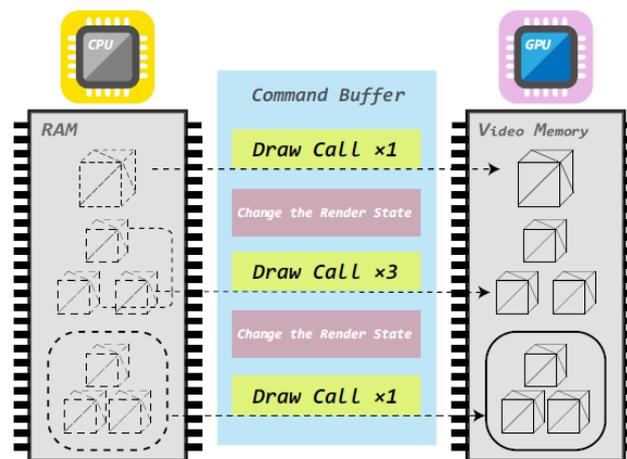


Figure 2. We use super object which is the combination of objects to reduce the frequency of “Draw Call” . When the rendering information of different objects is combined and packaged to generate a super object, they will use only one “Draw Call” of the super object.

Evaluation: Edge Bundling Comparison

In line rendering part, we modify the classic edge bundling algorithm by taking the semantic information of biomedical knowledge into consideration. For example, in the encephalic region scale, all the bundling directions should start from an encephalic region to another encephalic region. The original fibers rendered without any edge bundling algorithm, the bundled fibers generated by the modified algorithm by considering semantic information, and the further improved fibers by reducing visual clutter can be seen in Figure 3 (a), Figure 3 (b) and Figure 3 (c), respectively. The domain experts really appreciate the bundling effect customized by ScaleTraversal.

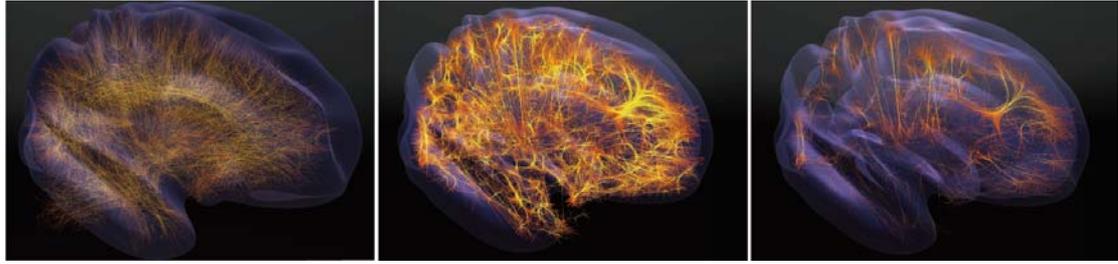


Figure 3. The comparison results of the line rendering approaches by using the fiber scale data. (a) The original fibers rendered without edge bundling. (b) The bundled fibers generated by the modified edge bundling algorithm. We modify the edge bundling algorithm [19] by taking the semantic information of biomedical knowledge into consideration. (c) The visual clutter is further reduced by querying the fibers between encephalic regions.

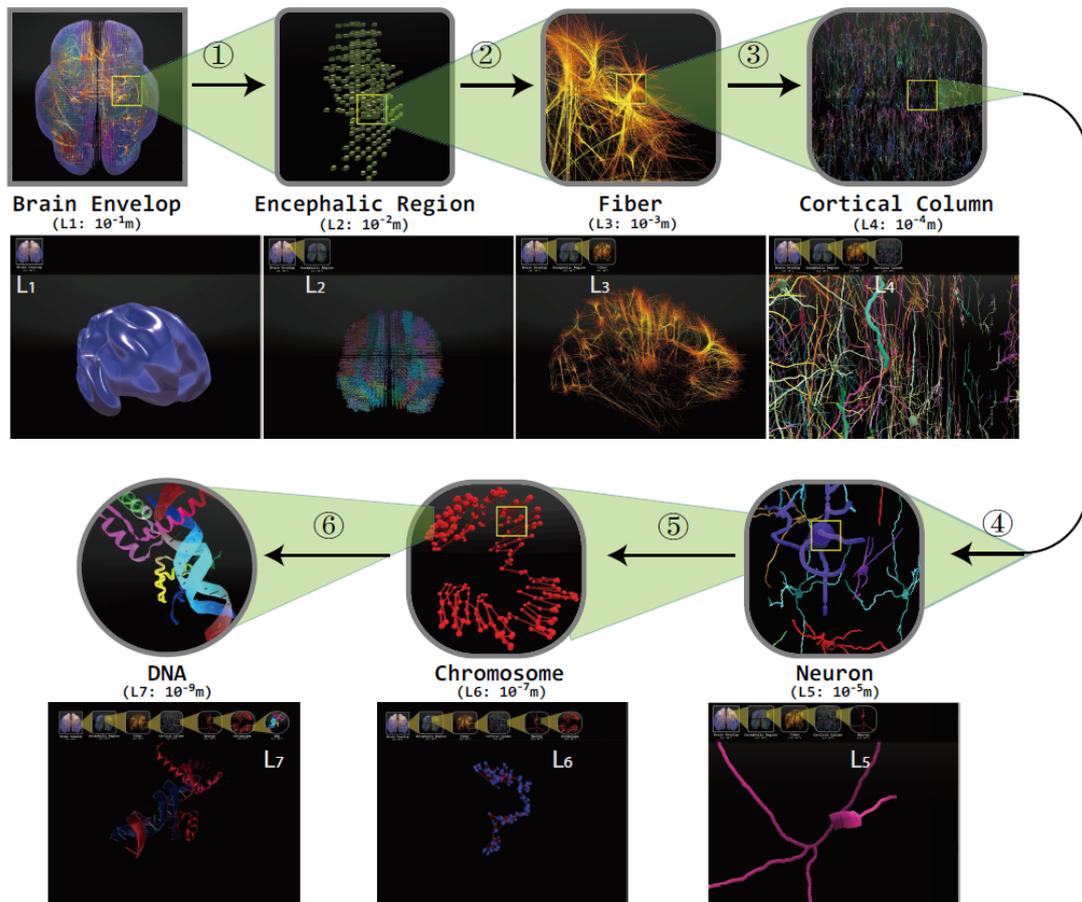


Figure 4. The third animation customization case by using all seven scales step by step. It includes brain envelop scale (L1), encephalic region scale (L2), fiber scale (L3), cortical column scale (L4), neuron scale (L5), chromosome scale (L6) and DNA scale (L7). The top icons are visualized in the animation space indicating which scale and where the current position the virtual camera are locating. The bottom sub-figures are the corresponding snapshots.

Animation Example: The Third Animation Example

The third case demonstrates the customized data animation across all the scales step by step, as shown in Figure 4. It includes brain envelop scale (L1), encephalic region scale (L2), fiber scale (L3), cortical column scale (L4), neuron scale (L5), chromosome scale (L6) and DNA scale (L7). The customization process is similar to the video edit process in Adobe Premiere. For more details about the second animation, please see the supplementary video of the submission.

DSL-based Short Script Codes Design

We have designed an interactive bi-functional user interface to customize multi-scale biomedical demonstration animations intuitively. It consists of a graphical (i.e, GUI controls) and a textual grammar (simple DSL codes). They are fully utilized to strengths of GUI's user friendliness and textual grammar's flexibility. The DSL-based short script codes will be embedded into the GUI interface of ScaleTraversal.

DSL codes samples:

```
setSurfaceStyle {metallic=float, smoothness=float, rgba=(float, float, float, float);
```

Description: customize the surface style of the rendering, set the metallic and smoothness of brain envelope

```
setLineStyle {maxr=float, minr=float, bundling=boolean, rgba=(float, float, float, float);
```

Description: customize the line style of the rendering, set the maximum and minimum radius and choose whether to binding. The line bundling effect is shown in Figure 5.

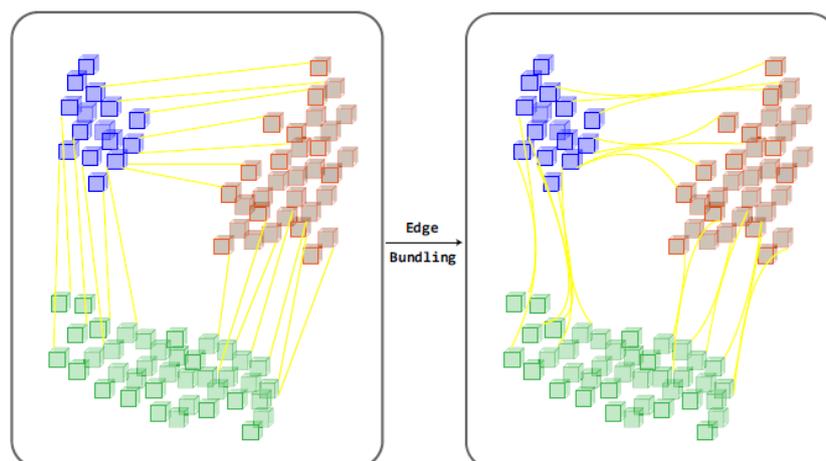


Figure 5. The illustration of edge bundling for the lines present in multi-scale biomedical data. We take the semantic information of biomedical knowledge into consideration. All the bundling directions should start from an encephalic region to another encephalic region.

setPointSize {maxr=**float**, minr=**float**, rgba=(**float**, **float**, **float**, **float**)};

Description: customize the point style of the rendering, set the size of each point

fadeInScale {scale=**string**, duration=**float** seconds, interval=**float** seconds};

Description: access to deeper scale in animation transition by setting the level, duration and interval

fadeOutScale {scale=**string**, duration=**float** seconds, interval=**float** seconds};

Description: back to superficial level in customized animation transition

drawPoints (scale=**string**, at=**string**, sampling-rate=**float**)

Description: draw point objects at a given functional area of the data in a specific scale. Lines are filtered with a given sampling-rate.

drawLines (scale=**string**, at=**string**, sampling-rate= **float**)

Description: draw lines at a given functional area of the data in a specific scale. Lines are filtered with a given sampling-rate.

drawSurface (scale=**string**, at=**string**, sampling-rate= **float**)

Description: draw surface objects at a given functional area of the data in a specific scale. Lines are filtered with a given sampling-rate.

translate {position=**string*****float** ,duration=**float** seconds, interval=**float** seconds};

Description: shift the camera lens across an interesting object or a complete scale by the given direction and length

rotate {centre=**string**, axis=**string**, angle=**float** degrees, ,duration=**float** seconds, interval=**float** seconds};

zoom {times=**float**, duration=**float** seconds, interval=**float** seconds};

Description: zoom-in/zoom-out the camera lens

parallel {

translate {direction=**string**, displacement=**float** ,duration=**float** seconds, interval=**float** seconds}=true/false;

rotate {centre=**string**, axis=**string**, angle=**float** degrees, ,duration=**float** seconds, interval=**float** seconds}=true/false;

scale {times=**float**, duration=**float** seconds, interval=**float** seconds}=true/false;

};

Description: executed concurrently.

Description: rotate the camera lens around an interesting object or a complete scale by the give centre, axis and angle. The parameter centre can be the centre of any of the brain function areas.

There are 92 brain functional areas in total in the multi-scale brain data provided by the domain experts. The names of functional areas are parameters customized by GUI interface. Users just need to input the partial letters by fuzzy search:

Precentral_L
Precentral_R
Frontal_Sup_L
Frontal_Sup_R
Frontal_Sup_Orb_L
Frontal_Sup_Orb_R
Frontal_Mid_L
Frontal_Mid_R
Frontal_Mid_Orb_L
Frontal_Mid_Orb_R
Frontal_Inf_Oper_L
Frontal_Inf_Oper_R
Frontal_Inf_Tri_L
Frontal_Inf_Tri_R
Frontal_Inf_Orb_L
Frontal_Inf_Orb_R
Rolandic_Oper_L
Rolandic_Oper_R
Supp_Motor_Area_L
Supp_Motor_Area_R
Olfactory_L
Olfactory_R
Frontal_Sup_Med_L
Frontal_Sup_Med_R
Frontal_Med_Orb_L
Frontal_Med_Orb_R
Rectus_L
Rectus_R
Insula_L
Insula_R
Cingulum_Ant_L
Cingulum_Ant_R
Cingulum_Mid_L
Cingulum_Mid_R
Cingulum_Post_L
Cingulum_Post_R
Hippocampus_L

Hippocampus_R
ParaHippocampal_L
ParaHippocampal_R
Amygdala_L
Amygdala_R
Calcarine_L
Calcarine_R
Cuneus_L
Cuneus_R
Lingual_L
Lingual_R
Occipital_Sup_L
Occipital_Sup_R
Occipital_Mid_L
Occipital_Mid_R
Occipital_Inf_L
Occipital_Inf_R
Fusiform_L
Fusiform_R
Postcentral_L
Postcentral_R
Parietal_Sup_L
Parietal_Sup_R
Parietal_Inf_L
Parietal_Inf_R
SupraMarginal_L
SupraMarginal_R
Angular_L
Angular_R
Precuneus_L
Precuneus_R
Paracentral_Lob_L
Paracentral_Lob_R
Caudate_L
Caudate_R
Putamen_L
Putamen_R
Pallidum_L
Pallidum_R
Thalamus_L
Thalamus_R
Heschl_L
Heschl_R
Temporal_Sup_L

Temporal_Sup_R
Temporal_Pole_Sup_L
Temporal_Pole_Sup_R
Temporal_Mid_L
Temporal_Mid_R
Temporal_Pol_Mid_L
Temporal_Pol_Mid_R
Temporal_Inf_L
Temporal_Inf_R
LGN_L
LGN_R