Projects for
Chapter 4: The Visualization Pipeline

1 PROJECT 1

Implement a simple data representation for a directed acyclic dataflow graph in terms of nodes which have an $update()$ function that prints the node's name. Separately, implement a $connect(n_1, n_2)$ function that, given a node n_1 and another node n_2, creates a directed edge from n_1 to n_2. Finally, given a user-specified start node n_S in this graph, implement a global procedure that updates the subgraph depending on the start node. By this we mean that:

- All nodes which depend, directly or indirectly, on the start node n_S should be updated;
- A node gets updated only when all the nodes it depends on, directly or indirectly, have been updated;
- Any node gets updated at most once during the above process.

Test your implementation by constructing a graph of moderate size (*e.g.* 15..20 nodes), letting the user specify a start node, and next printing the order in which nodes get updated.

2 PROJECT 2

Extend the simple dataflow-graph implementation described in Project 1 to support computational nodes which have multiple typed *inputs* and *outputs*. By this, we mean that a node

- has one or several (named) inputs and outputs;

- connections between nodes are realized between their respective inputs and outputs;
- besides a *name* (used to identify an input or output), an input or output also has a *type*, that specifies what kind of data the input accepts and the output provides, respectively.

Implement a simple scheme that would allow only *compatible* inputs and outputs to be connected. That is, an input having a type T_i can be connected to an output having a type T_o if and only if T_o is compatible with T_i, *i.e.* a data value of type T_o can be written to a data variable having type T_i.

Test your result in the same way as for Project 1, by building a simple dataflow graph, updating it from a given start node, and printing the order of executed update actions.

Hints:

- Model input and output types by actual data-types in your programming language, so that compatibility is automatically checked by your language's compiler or interpreter. For this, you would need to use a strongly typed programming language.
- For simplicity, assume that an input can only be connected to a single output at a given time. However, an output could be connected to more inputs at the same time.

3 PROJECT 3

Extend your implementation of the visualization pipeline discussed for Projects 1 and/or 2 to implement the *back-tracing* functionality outlined at Exercise 7 (Chapter 4).

To actually have your visualization pipeline create a (simple) visualization, implement

- a single dataset, *e.g.* a 2D uniform grid with scalar data,
- a few simple operations, *e.g.* doing simple math on the vertex scalar values, such as multiplying them with a constant or adding two datasets which have the same grid sizes,
- a simple rendering operation such as the height plot described in Chapter 2.

To test your program, implement a simple cell or vertex selection mechanism, either based on vertex or cell index-coordinates given from the command line or a GUI, or based on direct mouse-based selection in the rendering window (using GLUT mouse callbacks). Show the results of 'back tracing' computations for this cell as text on the standard output.

End of Projects for Chapter 4: The Visualization Pipeline