Pulling Visitors

Inverting Visitor-Based Control Flow

Agenda

Daniel J H, works for Mapbox on Graphs

Boost.Graph introduction, from visitors to iterators

https://github.com/daniel-j-h/cppnow2016 (git.io/cppnow2016-bgl)


San Pablo Bay
Wildlife Area
Richmond
Tiburon
Berkeley
Walnut Creek
Clayton
Concord
Pleasant Hill
Martinez
Benicia
Mount Diablo State
Park
Blackhawk
Danville
San Ramon
Dublin
Pleasanton
Livermore
San Francisco
Pacifica
Hillsborough
San Mateo
Moss Beach

Boost.Graph’s Generic Building-Blocks

Data structures (graph types)

Iterators (edges, vertices)

Properties (internal, external)

Algorithms (breadth-first search, dijkstra)

Visitors (examine_vertex)

Graph Concepts

Graph types are models for Graph Concepts, determines functionality

IncidenceGraph (source, target, out_edges)

BidirectionalGraph (in_edges)

VertexListGraph (vertices)

Graph Representations

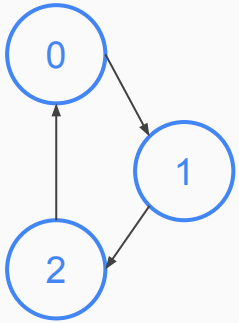
Customizable through template tags: Directed, Undirected, Properties

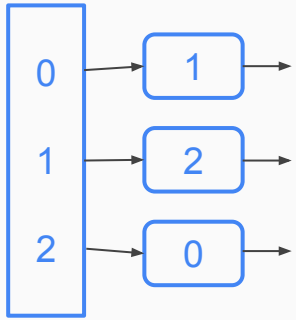
adjacency_list

adjacency_matrix

compressed_sparse_row_graph

adjacency_list<vecS, listS, directedS>

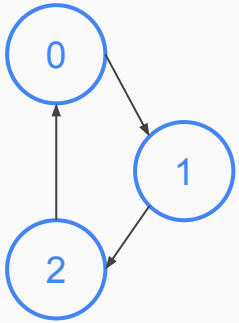


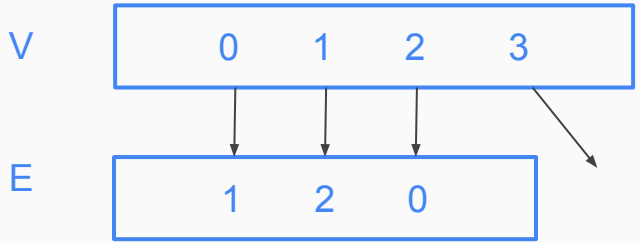


```
using graph_t = adjacency_list<vecS, vecS, directedS>;


graph_t graph(3);


add_edge(0, 1, graph);

add_edge(1, 2, graph);

add_edge(2, 0, graph);
```

```
struct edge_data_t { int duration = 0; };

using graph_t = adjacency_list<vecS, vecS, directedS, no_property, edge_data_t>;

graph_t graph(3);

add_edge(0, 1, edge_data_t{100}, graph);


auto duration = [&graph](auto edge) { return graph[edge].duration; };

auto positive = [](auto duration) { return duration > 0; };


auto d = accumulate(edges(graph) | transformed(duration) | filtered(positive), 0);
```

compressed_sparse_row_graph<directedS>






```
edges(v):
  first = E[V[v]];
  last  = E[V[v + 1]];
```

```
using graph_t = compressed_sparse_row_graph<directedS>;

using vertex_t = graph_traits<graph_t>::vertex_descriptor;


vector<vertex_t> sources{0, 1, 2};

vector<vertex_t> targets{1, 2, 0};


auto tag = construct_inplace_from_sources_and_targets;

graph_t graph{tag, sources, targets, 3};
```

Graph Algorithms

Visitors provide algorithm customization points

Graph Walk (bfs, dfs)

Shortest Paths (dijkstra, a-star)

Max-Flow / Min-Cut (edmonds_karp_max_flow)

```
struct discover_visitor : default_bfs_visitor {
    void discover_vertex(const vertex_t vertex, const graph_t&) {
        cout << vertex << endl;
    }
};

vertex_t source{0};
breadth_first_search(graph, source, visitor(discover_visitor{}));
```

Use-Case Bidirectional Dijkstra

Baseline router to compare against

Start first search on graph from source

Start second search on reversed graph from target

Step both searches (ping-pong) until they meet in the middle

Problem: how to stop and resume visitors

```
vertex_t middle;


async(dijkstra_shortest_path(graph, source, visitor(ping_pong{middle});

async(dijkstra_shortest_path(rev_graph, target, visitor(ping_pong{middle});
```

Coroutines for Cooperative Multitasking

Bind coroutine to visitor, get lazy Dijkstra generator for free

No explicit synchronization, no threads (concurrency != parallelism)

Aha Moment: can be stopped, can be resumed, proper iterators (stdlib)

Technique works for all visitors, and especially well for Boost.Graph

```
using coro_t = coroutines::asymmetric_coroutine<vertex_t>;


struct dijkstra_stepwise : default_dijkstra_visitor {
    dijkstra_stepwise(coro_t::push_type& sink_) : sink(sink_) {}

    void examine_vertex(const vertex_t vertex, const graph_t&) const {
        sink(vertex);
    }
    coro_t::push_type& sink;
};
```

```
coro_t::pull_type lazy_forward_vertices{[&](auto& sink) {
    dijkstra_shortest_paths_no_color_map(graph, source,
        weight_map(get(&edge_data_t::distance, graph))
        .predecessor_map(forward_prev_map)
        .visitor(dijkstra_stepwise{sink}));
}};

while (lazy_forward_vertices && lazy_backward_vertices)
    // lazy_forward_vertices.get();  lazy_forward_vertices();
```

```
auto poi = [&graph](auto vertex) { return has_poi(vertex, graph); };

auto it = find_if(lazy_forward_vertices, poi);


if (it != end(lazy_forward_vertices))

    std::cout << *it << std::endl;
```

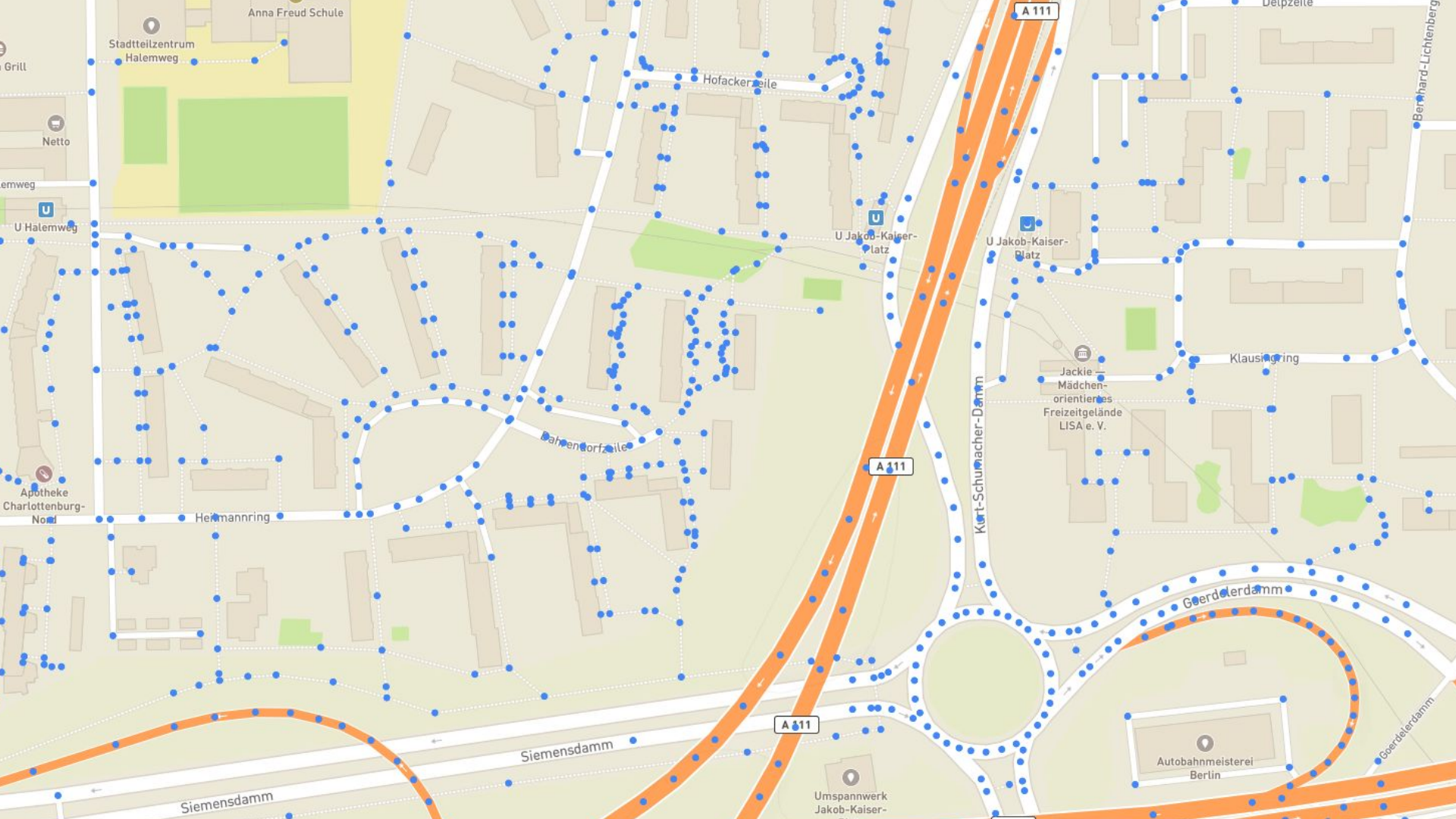
Give Boost.Graph a try!

OpenStreetMap (extract nodes and ways: libosmium)

Construct graph, add properties (location, Boost.Geometry Haversine distance)

Route on the graph (Boost.Geometry’s RTree for initial coordinate lookup)

Visualize the graph (simplification: tippecanoe)


A 111
Anna Freud Schule
Stadtteilzentrum Halemweg
Hofackerzeile
Netto
U Halemweg
U Jakob-Kaiser-Platz
U Jakob-Kaiser-Platz
Delpzeile
Bernhard-Lichtenberg
Klausenerring
Jackie – Mädchenorientiertes Freizeitgelände LISA e. V.
Kurt-Schumacher-Damm
Bahnendorfzeile
Apotheke Charlottenburg-Nord
Heilmannring
Goerdelerdamm
Siemensdamm
Siemensdamm
Autobahnmeisterei Berlin
Umspannwerk Jakob-Kaiser-

Our Take Aways

Powerful trio: Boost.Graph + Boost.Range + Boost.Geometry

Switching to 32 bit vertex and edge index types in CSR (size_t default)

Parallel Boost.Graph vs. r3.4xlarge (120 GB RAM), r3.8xlarge (250 GB RAM)

Boost.Graph + Boost.Coroutine: from visitors to generators, stdlib integration