CARLA适配异常xodr

之前的deqing.xodr导入carla会导致UE崩溃；

```
(base) crist@crist:~/homework$ python ./xodr_health_check.py map/deqing.xodr
==== XODR Sanity Report ====
Warnings (2):
  [WARN] header/geoReference is empty; CARLA 会使用默认值
  [WARN] junction(id=11543) 没有任何 connection

Errors (3):
  [FATAL] road(id=3)/elevation[7] s 非单调递增 (prev=90.95499936742635, now=90.95499861334935)
  [FATAL] road(id=35)/elevation[1] s 非单调递增 (prev=0.0, now=-4.7659579945e-06)
  [FATAL] road(id=35)/elevation[7] s 非单调递增 (prev=55.44205686194254, now=55.442055235922936)
(base) crist@crist:~/homework$
```

在CARLA导入xodr，会用多项式来计算路面每个采样点的高度。如果道路起点的距离不是单调递增，或者出现负数会导致找不到正确的段（例如 s=90.954999 之后却又出现 90.954998）或者 样条拼接错误，生成NaN崩溃。

修改点：

函数GenerateChunkedMesh()里加“空/退化/非有限”检查与早退，基本可以从根上避免 SIGSEGV。

丢弃空/NaN/Inf 的有问题的mesh；路口合并有空指针与空输入兜底；全链路都避免 NaN 和越界；参数值不至于极端。

修改位置：**Libarla/source/carla/road/Map.cpp**

代码块

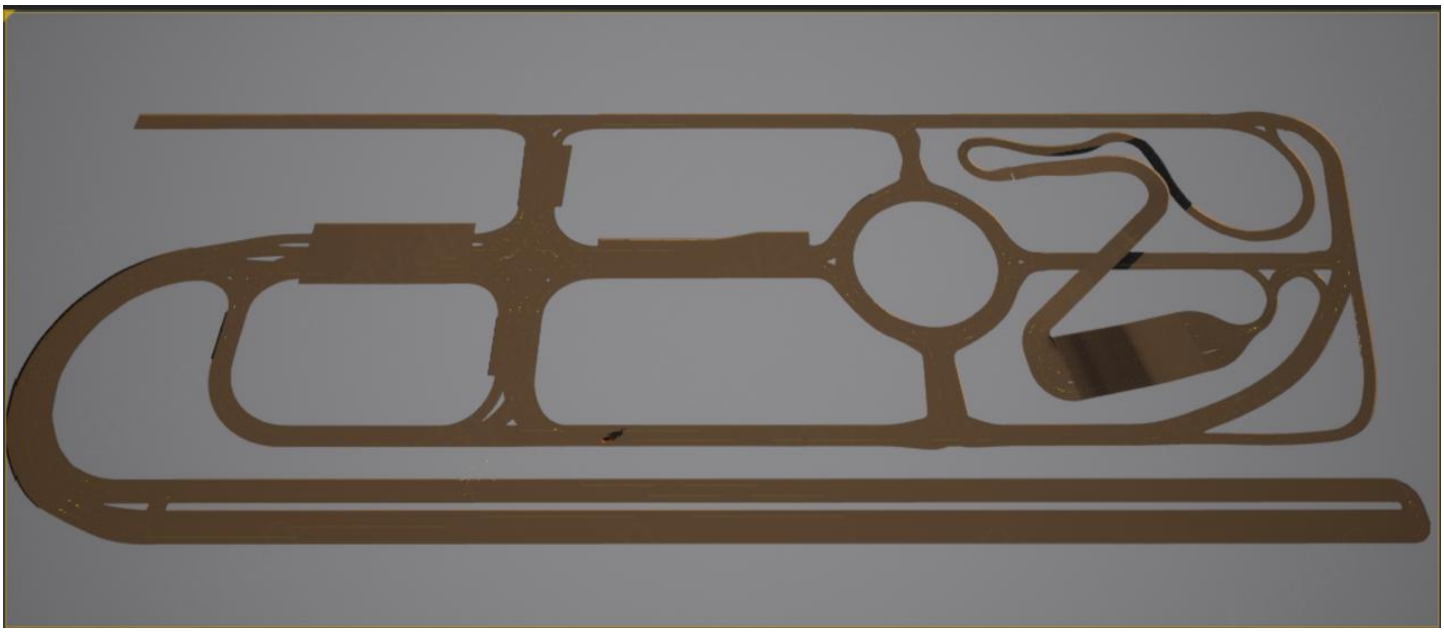
```
std::vector<std::unique_ptr<geom::Mesh>> Map::GenerateChunkedMesh(
    const rpc::OpendriveGenerationParameters& params) const {
  geom::MeshFactory mesh_factory(params);
  std::vector<std::unique_ptr<geom::Mesh>> out_mesh_list;

  // ---- Safety: clamp bad params to sane values ----
  const double kMinChunk = 20.0;     // meters
  const double kMaxChunk = 200.0;    // meters
  const double kFallbackChunk = 100.0;
  const double max_road_len =
      (std::isfinite(params.max_road_length) && params.max_road_length > 0.0)
        ? std::min(std::max(params.max_road_length, kMinChunk), kMaxChunk)
        : kFallbackChunk;

  // Helper lambdas
  auto mesh_valid = [](const std::unique_ptr<geom::Mesh>& m)->bool {
    if (!m) return false;
```

```
18        const auto &V = m->GetVertices();
19        const auto &I = m->GetIndexes();
20        if (V.empty() || I.empty()) return false;
21        // Finite check on a few sentinel vertices (avoid O(n) full scan)
22        auto finite = [](const geom::Vector3D &p){
23          return std::isfinite(p.x) && std::isfinite(p.y) && std::isfinite(p.z);
24        };
25        if (!finite(V.front())) return false;
26        if (!finite(V.back()))  return false;
27        return true;
28      };
29      auto try_add_mesh = [&](std::unique_ptr<geom::Mesh> m){
30        if (mesh_valid(m)) out_mesh_list.emplace_back(std::move(m));
31      };
32      std::unordered_map<JuncId, geom::Mesh> junction_map;
33      for (auto &&pair : _data.GetRoads()) {
34        const auto &road = pair.second;
35        if (!road.IsJunction()) {
36          std::vector<std::unique_ptr<geom::Mesh>> road_mesh_list =
    mesh_factory.GenerateAllWithMaxLen(road);
37          // Filter: drop null/empty/invalid meshes instead of bulk-inserting
    blindly
38          for (auto &m : road_mesh_list) {
39            try_add_mesh(std::move(m));
40          }
41        }
42      }
43
44      // Generate roads within junctions and smooth them
45      for (const auto &junc_pair : _data.GetJunctions()) {
46        const auto &junction = junc_pair.second;
47        std::vector<std::unique_ptr<geom::Mesh>> lane_meshes;
48        std::vector<std::unique_ptr<geom::Mesh>> sidewalk_lane_meshes;
49        for(const auto &connection_pair : junction.GetConnections()) {
50          const auto &connection = connection_pair.second;
51          const auto &road = _data.GetRoads().at(connection.connecting_road);
52          for (auto &&lane_section : road.GetLaneSections()) {
53            for (auto &&lane_pair : lane_section.GetLanes()) {
54              const auto &lane = lane_pair.second;
55              if (lane.GetType() != road::Lane::LaneType::Sidewalk) {
56                 auto m = mesh_factory.Generate(lane);
57                 if (mesh_valid(m)) lane_meshes.emplace_back(std::move(m));
58              } else {
59                 auto m = mesh_factory.Generate(lane);
60                 if (mesh_valid(m))
    sidewalk_lane_meshes.emplace_back(std::move(m));
61              }
```

```
            }
          }
        }
        if (lane_meshes.empty() && sidewalk_lane_meshes.empty())
          continue;

        if (params.smooth_junctions) {
          // MergeAndSmooth may assume non-empty input; guard it
          std::unique_ptr<geom::Mesh> merged_mesh;
          if (!lane_meshes.empty())
            merged_mesh = mesh_factory.MergeAndSmooth(lane_meshes);
          else
            merged_mesh = std::make_unique<geom::Mesh>();
          // merged_mesh might still be null if internal failed; guard
          if (!merged_mesh)
            merged_mesh = std::make_unique<geom::Mesh>();
          for (auto &lane : sidewalk_lane_meshes) {
            if (lane) *merged_mesh += *lane;
          }
          if (mesh_valid(merged_mesh))
            out_mesh_list.push_back(std::move(merged_mesh));
        } else {
          std::unique_ptr<geom::Mesh> junction_mesh =
std::make_unique<geom::Mesh>();
          for (auto &lane : lane_meshes) {
            if (lane) *junction_mesh += *lane;
          }
          for (auto &lane : sidewalk_lane_meshes) {
            if (lane) *junction_mesh += *lane;
          }
          if (mesh_valid(junction_mesh))
            out_mesh_list.push_back(std::move(junction_mesh));
        }
      }

      // If after filtering nothing remains, return empty safely
      if (out_mesh_list.empty())
        return {};

      // Find first valid vertex to seed bounds
      geom::Vector2D min_pos, max_pos;
      bool seeded = false;
      for (auto &mesh : out_mesh_list) {
        if (!mesh) continue;
        const auto &V = mesh->GetVertices();
        if (V.empty()) continue;
        const auto v = V.front();
```

```
108         if (!std::isfinite(v.x) || !std::isfinite(v.y)) continue;
109         min_pos = geom::Vector2D(v.x, v.y);
110         max_pos = min_pos;
111         seeded = true;
112         break;
113       }
114       if (!seeded) return {}; // nothing with finite vertices
115
116       // Expand bounds; use first and last vertex as cheap sentinels
117       for (auto &mesh : out_mesh_list) {
118         if (!mesh) continue;
119         const auto &V = mesh->GetVertices();
120         if (V.empty()) continue;
121         const auto v0 = V.front();
122         const auto v1 = V.back();
123         auto upd = [&](const geom::Vector3D &pt){
124           if (!std::isfinite(pt.x) || !std::isfinite(pt.y)) return;
125           min_pos.x = std::min(min_pos.x, pt.x);
126           min_pos.y = std::min(min_pos.y, pt.y);
127           max_pos.x = std::max(max_pos.x, pt.x);
128           max_pos.y = std::max(max_pos.y, pt.y);
129         };
130         upd(v0); upd(v1);
131       }
132
133       // Compute chunk grid with clamped max_road_len
134       const double span_x = std::max(0.0, static_cast<double>(max_pos.x -
    min_pos.x));
135       const double span_y = std::max(0.0, static_cast<double>(max_pos.y -
    min_pos.y));
136
137       const size_t mesh_amount_x = static_cast<size_t>(span_x / max_road_len) +
    1u;
138       const size_t mesh_amount_y = static_cast<size_t>(span_y / max_road_len) +
    1u;
139       std::vector<std::unique_ptr<geom::Mesh>> result;
140       if (mesh_amount_x == 0 || mesh_amount_y == 0)
141         return {};
142       const size_t grid_n = mesh_amount_x * mesh_amount_y;
143       result.reserve(grid_n);
144       for (size_t i = 0; i < grid_n; ++i) {
145         result.emplace_back(std::make_unique<geom::Mesh>());
146       }
147       for (auto &mesh : out_mesh_list) {
148         if (!mesh) continue;
149         const auto &V = mesh->GetVertices();
150         if (V.empty()) continue;
```

```
151         const auto v = V.front();
152         if (!std::isfinite(v.x) || !std::isfinite(v.y)) continue;
153         size_t x_pos = static_cast<size_t>((v.x - min_pos.x) / max_road_len);
154         size_t y_pos = static_cast<size_t>((v.y - min_pos.y) / max_road_len);
155         // Guard: clamp to grid
156         if (x_pos >= mesh_amount_x) x_pos = mesh_amount_x - 1;
157         if (y_pos >= mesh_amount_y) y_pos = mesh_amount_y - 1;
158         const size_t idx = x_pos + mesh_amount_x * y_pos;
159         if (idx < result.size()) {
160           *(result[idx]) += *mesh;
161         }
162       }
163 
164       return result;
165     }
```



修改后即可正常导入。