Traverse: An iOS App to Navigation Campuses

Rashid Almheiri & Younis Almazrooqi

2025-04-22

Abstract

Traverse is an innovative iOS application designed to facilitate indoor navigation within the American University in Dubai (AUD) campus. Utilizing Apple's Indoor Mapping Data Format (IMDF), it offers detailed campus maps, accessibility information, and real-time pathfinding. The system integrates a high-performance C++ backend for graph-based algorithms with a user-friendly SwiftUI frontend. This project demonstrates the effective combination of advanced mapping technologies and efficient pathfinding to enhance navigation in educational settings.

Contents

Introduction

Traverse is an iOS indoor navigation application developed for the American University in Dubai (AUD). It leverages the Indoor Mapping Data Format (IMDF) to provide detailed campus navigation, accessibility information, and real-time pathfinding for students and visitors. The project demonstrates a modular approach, combining a performant C++ backend for graph-based algorithms with a modern SwiftUI frontend for iOS.

Project Overview

Traverse helps users navigate complex indoor environments, such as university campuses. The application loads IMDF data describing the campus, including buildings, levels, rooms, and amenities. Users can search for locations, view accessibility information, and receive step-by-step navigation instructions. The system is built with extensibility in mind, allowing support for additional campuses and features in the future.

Application Interface

The Traverse application features a modern, intuitive interface designed for easy navigation:

Figure 1: Main application interface showing the AUD campus map with building overlays

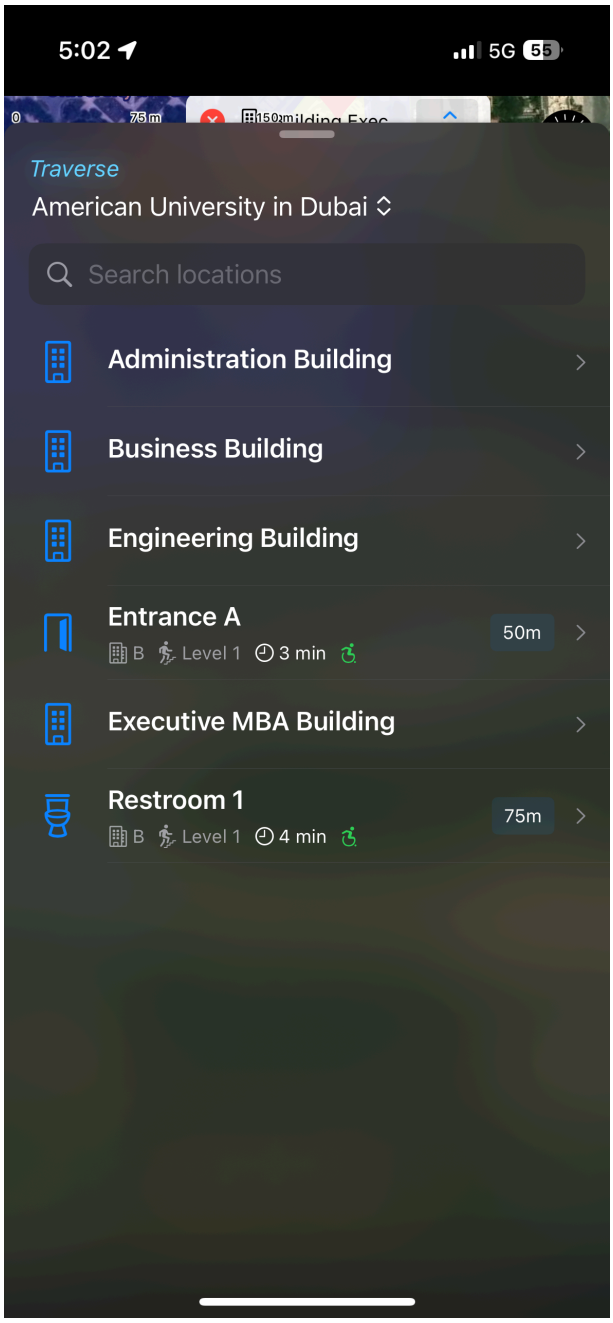

Figure 2: Search interface with real-time location suggestions

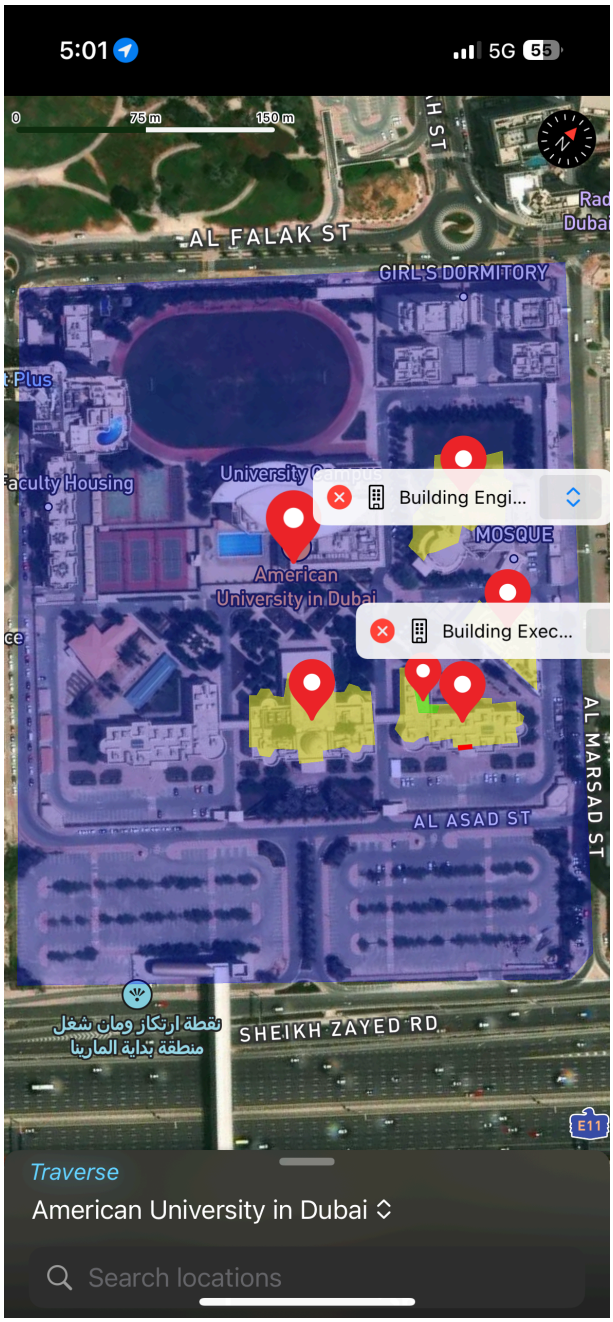

Figure 3: Active navigation showing the shortest path between two locations

IMDF Data Structure

IMDF (Indoor Mapping Data Format) is Apple's open standard for indoor mapping. For the AUD campus implementation, we utilize the following key components:

Core IMDF Components

- **Venue**: Represents the entire AUD campus as a top-level container
- **Buildings**: Individual structures within the campus
- **Levels**: Floor plans within buildings

- **Units**: Rooms, corridors, and other indoor spaces
- **Amenities**: Points of interest like restrooms, water fountains, etc.
- **Openings**: Doors, stairs, and other transition points
- **Relationships**: Hierarchical connections between different elements

IMDF File Structure

Our implementation uses a collection of GeoJSON files:

```
└── data/imdf/AUD.imdf/
    ├── manifest.json
    ├── venue.json
    ├── building.json
    ├── level.json
    ├── unit.json
    ├── opening.json
    └── amenity.json
```

Graph Construction from IMDF

The navigation graph is constructed by:

1. Loading and parsing IMDF files using `IMDFDecoder`
2. Creating nodes for each navigable point (unit centers, openings)
3. Establishing edges between connected spaces
4. Calculating weights based on real-world distances
5. Adding accessibility metadata to edges

Internal Architecture

C++ Graph and Pathfinding

The core pathfinding logic is implemented in C++ for performance and portability. The `Graph` class represents the campus as a weighted undirected graph, where nodes are locations (e.g., rooms, entrances) and edges represent navigable paths with associated weights (distances or costs). Dijkstra's algorithm computes the shortest path between any two nodes.

Swift IMDF Model

The Swift codebase defines models for IMDF features (Venue, Building, Level, Unit, etc.). The `IMDFDecoder` loads and parses the IMDF dataset, mapping GeoJSON features to Swift objects. Relationships between features (e.g., which units are on which level) are maintained for efficient querying and navigation.

C++/Swift Bridge

A planned bridging layer will expose the C++ pathfinding logic to Swift, allowing the iOS app to request shortest paths by passing node identifiers to the C++ backend and receiving the computed route. This design supports future platform expansions (e.g., Android) using the same C++ core.

SwiftUI Frontend

The user interface, built with SwiftUI, provides a modern, responsive experience. Main components include:

- **Map View**: Displays the campus map, buildings, and navigation routes.
- **Search Bar**: Enables location and amenity searches.
- **Detail Views**: Show location details, including accessibility and peak times.
- **Navigation Controls**: Allow users to start navigation and view step-by-step routes.

Data Flow and Internal Workings

1. **IMDF Loading**: On launch, the app loads the IMDF dataset for the selected university. The `IMDFDecoder` parses GeoJSON files into Swift model objects.
2. **Graph Construction**: The C++ `Graph` is built from IMDF relationships, with nodes for navigable locations and edges for connections.
3. **User Interaction**: Users search for or select a location, and the app identifies the corresponding graph node.
4. **Pathfinding**: Navigation requests trigger the C++ backend (via the bridge) to compute the shortest path.
5. **Route Visualization**: The computed path is displayed on the map with step-by-step instructions and accessibility details.

User Manual

Getting Started

1. Install and launch the Traverse app on your iOS device.
2. Select the AUD campus on first launch (currently the only supported campus).
3. View the campus layout in the main map view.

Searching for Locations

1. Use the top search bar to find rooms, amenities, or buildings.
2. Tap a search result for details, including accessibility and peak times.

Navigating the Campus

1. Select a location and tap **Go** to start navigation.
2. Follow the highlighted shortest path on the map.
3. The app updates progress in real time.

Accessibility Features

- Displays accessibility details (e.g., wheelchair access, braille signage).
- Prioritizes accessible routes when possible.

Troubleshooting

- If the map fails to load, ensure IMDF data is in the app bundle.
- For navigation issues, verify device location services are enabled.

Extending Traverse

Adding a New Campus

1. Obtain or generate an IMDF dataset for the new campus.
2. Add IMDF files to the app's data directory.
3. Update the `University` model in Swift to include the new campus.
4. Rebuild the app to make the campus selectable.

Adding New Features

- Extend Swift model classes for new IMDF feature types.
- Update the UI to display new information or controls.
- Enhance the C++ `Graph` class and bridge for new pathfinding logic.

Code Structure

- `src/Graph.hh, Graph.cc` : C++ graph and pathfinding logic (Dijkstra's algorithm, node/edge management)
- `src/Bridge.hh` : Planned C++/Swift bridge interface
- `src/ios/Model/IMDF/` : Swift models for IMDF features (Venue, Building, Level, etc.)
- `src/ios/Model/IMDF/IMDFDecoder.swift` : Loads and parses IMDF data
- `src/ios/View/` : SwiftUI views for the UI (map, search, detail, navigation)
- `src/ios/ViewModel/` : View models for app state and business logic
- `src/ios/Extension/Geometry.swift` : Geometry helpers for map rendering

Graph Implementation

C++: Graph Class and Dijkstra's Algorithm

```cpp
// src/Graph.hh
#include <vector>
#include <string>
#include <unordered_map>

class Graph {
public:
    Graph();
    ~Graph();
    void addNode(const std::string& node);
    void addEdge(const std::string& from, const std::string& to, double weight);
    std::vector<std::string> shortestPath(const std::string& start, const std::string& end);
private:
    struct Edge {
        std::string to;
        double weight;
    };
    std::unordered_map<std::string, std::vector<Edge>> adjList;
};
```

```cpp
// src/Graph.cc (Dijkstra's algorithm)
std::vector<std::string> Graph::shortestPath(const std::string &start,
                                             const std::string &end) {
    if (adjList.find(start) == adjList.end() ||
        adjList.find(end) == adjList.end()) {
        return {};
    }
    std::unordered_map<std::string, double> distances;
    std::unordered_map<std::string, std::string> previous;
    for (const auto &pair : adjList) {
        distances[pair.first] = std::numeric_limits<double>::infinity();
    }
    distances[start] = 0;
    using P = std::pair<double, std::string>;
    std::priority_queue<P, std::vector<P>, std::greater<P>> pq;
    pq.push({0, start});
    while (!pq.empty()) {
        auto [dist, current] = pq.top();
        pq.pop();
        if (current == end) break;
        if (dist > distances[current]) continue;
```

```
        for (const auto &edge : adjList[current]) {
            double newDist = dist + edge.weight;
            if (newDist < distances[edge.to]) {
                distances[edge.to] = newDist;
                previous[edge.to] = current;
                pq.push({newDist, edge.to});
            }
        }
    }
    if (distances[end] == std::numeric_limits<double>::infinity()) return {};
    std::vector<std::string> path;
    std::string current = end;
    while (current != start) {
        path.push_back(current);
        current = previous[current];
    }
    path.push_back(start);
    std::reverse(path.begin(), path.end());
    return path;
}
```

Explanation: The C++ code defines a weighted undirected graph and implements Dijkstra's algorithm to find the shortest path, forming the core of the navigation logic.

Swift: IMDF Model Parsing and Map Rendering

```swift
// src/ios/Model/IMDF/IMDFDecoder.swift
class IMDFDecoder {
    let decoder = JSONDecoder()
    func decode(_ imdfDirectory: URL) throws -> DecodedIMDF {
        let archive = IMDFArchive(directory: imdfDirectory)
        // ...parsing logic for each IMDF feature...
        return DecodedIMDF(
            manifest: manifest,
            venue: venue,
            addresses: addresses,
            levels: levels,
            units: units,
            footprints: footprints,
            relationships: relationships,
            anchors: anchors,
            sections: sections,
            buildings: buildings,
            occupants: occupants,
            amenities: amenities,
```

```
            openings: openings
        )
    }
}
```

```swift
// src/ios/View/Component/VenueView.swift
struct VenueView: View {
    @State private var viewModel: VenueViewModel
    @Binding private var viewport: Viewport
    var body: some View {
        Map(viewport: $viewport) {
            ForEvery(viewModel.buildings, id: \.id) { building in
                if let polygon = building.geometry?.asPolygon {
                    PolygonAnnotation(
                        id: building.id.uuidString,
                        polygon: polygon
                    )
                    .fillColor(.yellow)
                    .fillOpacity(0.5)
                }
            }
            // ...other map layers...
        }
        .mapStyle(.standardSatellite(lightPreset: .day))
        .onAppear { viewModel.loadMap() }
    }
}
```

Explanation: The Swift code parses IMDF data into model objects and renders the map using SwiftUI, displaying buildings as polygons with interactive UI updates.

Conclusion

Traverse successfully addresses the need for efficient indoor navigation within the AUD campus. By leveraging IMDF and a robust C++ graph-based pathfinding algorithm, it provides accurate and accessible navigation. The SwiftUI frontend ensures a user-friendly interface, making campus navigation seamless for students and visitors. This project highlights the power of integrating advanced mapping and computational technologies to solve real-world challenges in educational settings.

Future Work

- Complete the C++/Swift bridge for native pathfinding
- Add support for more universities and IMDF datasets

- Enhance accessibility and real-time data integration