Route Reroute

Diogo Carvalho - 113221

Introduction to Computer Graphics – 2024/2025

Main ideas

What is the project?

- 3D driving game with sequential missions
- Time rewind mechanics for recording/replaying car movements
- 4 levels with progressive difficulty
- Achievement system and sandbox mode

Three.js usage

- Core Three.js library
- GLTF loader for 3D models
- Web Audio API integration
- No additional Three.js modules

What can players do?

- Drive different vehicles through urban environments
- Rewind time to replay previous movements
- Complete missions and unlock achievements
- Explore in sandbox mode with day/night cycles

Project Links

- GitHub repository (includes YouTube demo in README):

https://github.com/diogotavc/route-reroute

- Github Pages:

https://diogotavc.github.io/route-reroute

Models and the scene graph

Scene Organisation

- Root scene with grouped elements
- Map tiles for road construction
- Dynamic vehicle instances
- Lighting and UI overlays

Technical Features

- Instance cloning for performance
- OBB collision detection using SAT (Separating Axis Theorem)
- Hierarchical transforms

Asset Sources

- Kenney Asset Packs for all 3D models
- 20+ vehicles: cars, trucks, emergency vehicles
- Modular road system with intersections, crossroads, etc
- Urban environment: buildings, streetlights, props

```
Scene
├── Map Group
│   ├── Road Tiles (straight, bend, intersection, crossroad)
│   ├── Buildings (residential, commercial)
│   ├── Random Objects (trees, barriers, cones)
│   └── Street Lights (curved, square)
├── Car Models (20 different vehicles)
├── Lighting System
└── UI Overlays
```

Animation

Physics Movement

- Realistic car dynamics: acceleration, braking, steering

- Collision response using SAT algorithm with OBB detection

- Terrain effects for different surfaces

Time Mechanics

- Movement recording system

- Smooth interpolation during rewind

- Multi-car replay for traffic scenarios

Camera Effects

- Follow camera with smooth transitions

- Cinematic idle mode

- Firefly companion animations

UI Animations

- CSS transitions for overlays and menus

- Smooth hover effects and notifications

- Achievement popup animations

Illumination

Light Types Implementation

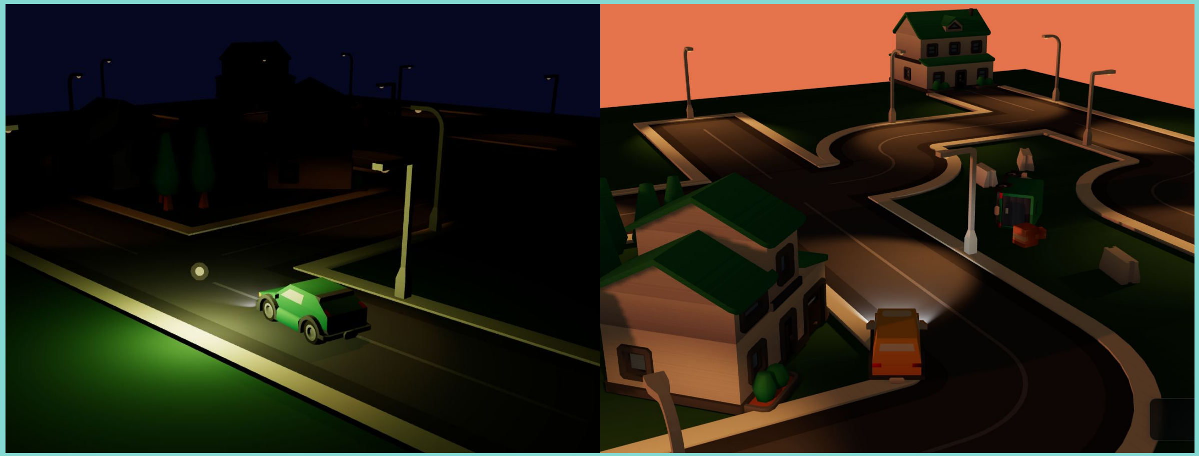
- **DirectionalLight**: Dynamic sun/moon positioning
- **AmbientLight**: Base illumination varying by time
- **SpotLight**: Car headlights and streetlights
- **PointLight**: Firefly effects and decorative lighting

Dynamic Features

- Automatic day/night cycle
- Dynamic shadows with quality settings
- Car headlight automation
- Street light grid activation

Advanced Effects

- Shadow mapping with PCF filtering
- Colour temperature shifts: cooler blues at dawn/dusk, warm oranges during day/night transitions
- Graphics quality presets



User Interaction

Keyboard Controls

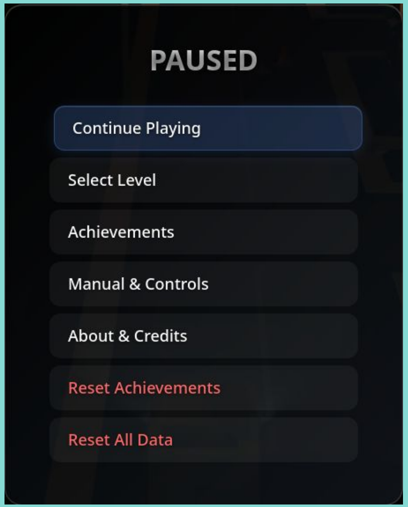
- WASD/Arrow keys for vehicle movement
- R key for time rewind and level restart with visual feedback
- C key for camera mode switching
- M, [,] for music system controls
- ESC, P for pause menu navigation

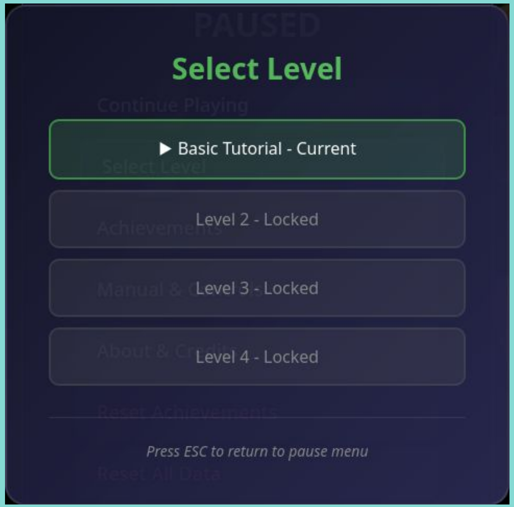
Mouse Controls

- Menu navigation with hover effects

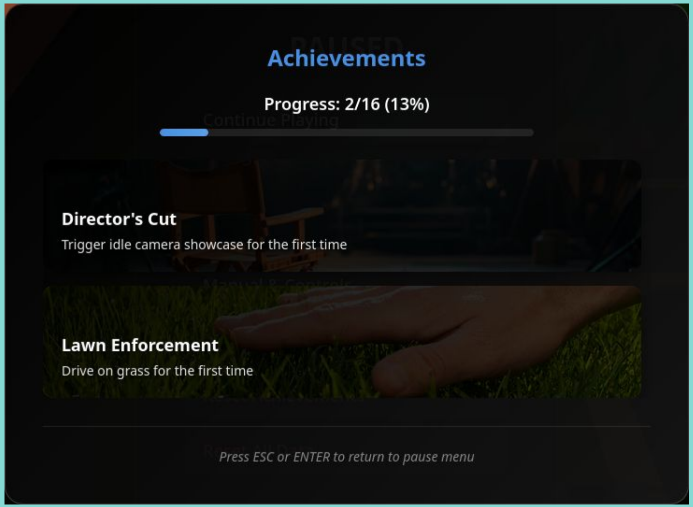
Interface Features

- Real-time HUD: speed, health, timer, mission information
- Achievement system with animated notifications
- Comprehensive pause menu system with smooth transitions





Press ESC to return to pause menu



Progress: 2/16 (13%)

Director's Cut
Trigger idle camera showcase for the first time

Lawn Enforcement
Drive on grass for the first time

Press ESC or ENTER to return to pause menu

Development

Architecture Overview

- Dedicated JavaScript files with clear responsibilities
- Modular structure with separated concerns
- Event-driven achievement system
- Configuration-driven gameplay parameters

Key Files and Responsibilities

- **carPhysics.js** Collision detection and movement
- **cars.js** Vehicle management and state
- **lights.js** Day/night cycle and lighting
- **interface.js** UI components and menus
- **achievements.js** Progress tracking system
- **mapLoader.js** Map layout creation

Problems and Solutions

- **Performance optimisation** Frustum culling, LOD distance thresholds, instance cloning, resolution scaling

- **Complex state management** Centralised configuration system

- **Browser compatibility** Graphics presets with graceful degradation

- **Physics stability** SAT collision detection with OBB for accurate vehicle collisions

Conclusions

Technical Achievements

- Complete 3D game engine built on Three.js
- Complex time manipulation mechanics
- Comprehensive lighting and graphics systems
- Scalable modular architecture

Learning Outcomes

- WebGL pipeline understanding through Three.js
- Real-time physics implementation with SAT collision detection
- User experience design for complex systems
- 3D graphics performance optimization: frustum culling, LOD management, shadow mapping

Future Enhancements

- Godot Engine remake
- Enhanced physics simulation
- Multiplayer capabilities
- Mobile platform support

References

Technical Documentation

- **Three.js Documentation** 3D rendering engine
- **Web Audio API Specification** Music system
- **GLTF 2.0 Specification** 3D model formats

Development Resources

- **MDN Web Docs** JavaScript APIs
- **Stack Overflow** Community solutions
- **WebGL Fundamentals** Graphics programming

Assets and Resources

- **Kenney Asset Packs** 3D models and textures
- **Gran Turismo Soundtrack** Background music
- **YouTube** Honking sound
- **Google Images** Achievement images