

Documentation

Group/Game Name: Apex Racing

Brief description of implementation:

As a foundation, we utilized the PTV framework from TUWEL and adapted it to our specific needs. We implemented custom Object classes for each of our entities (car, track, terrain, braking boards, etc.) and rendered them using the approach provided by the example framework. All 3D objects were modeled by us in Blender and are loaded into the Vulkan API using TinyGLTF. Additionally, we introduced a custom Texture Loader to map textures to their corresponding objects via a texture array.

Regarding the car's logic, we emulated several internal components (engine, gearbox, differential) ourselves to build a virtual vehicle that enhances gameplay through manual shifting and dedicated internal logic. Consequently, the car's driving behavior changes dynamically based on speed, gear, and external factors, imitating a real open-wheel race car. This is complemented by the Bullet Physics Engine, which handles all collision-based events, such as triggering new laps, detecting track limits, and colliding with static or dynamic boundary objects like the foam braking boards or the track barriers.

The world surrounding the racetrack is modeled using a heightmap, which is tessellated by introducing four different shader stages: a custom Vertex Shader, a Tessellation Control Shader (to calculate subdivision levels), a Tessellation Evaluation Shader (to sample the heightmap and displace vertices), and finally the Fragment Shader. The heightmap is passed as a grayscale PNG texture and was created by compositing various heightmaps in Photoshop to suit our environment.

For lighting and shading, the entire scene is illuminated by a directional sun using a Phong reflection model. We implemented Cascaded Shadow Mapping with Percentage-Closer Filtering (PCF) to render smooth, high-fidelity dynamic shadows for moving entities, alongside a static, baked shadow map for the terrain to save computing power. Adding to the shading pipeline, Environment Mapping is introduced to provide realistic reflections on the car's metallic surfaces, significantly enhancing the visual fidelity of the vehicle. Finally, for the user interface, we used the ImGui library included in the framework to build a simple HUD that is tied into the game loop to display real-time telemetry and visual feedback for track limits.

Additional libraries:

- Bullet Physics Engine for Collision Detection
<https://github.com/bulletphysics/bullet3>
- Tiny GLTF library used for extracting vertices, indices, normals and uvs from Blender .glb files
<https://github.com/syoyo/tinygltf.git>

Gameplay:

Mandatory:

- 3D Geometry: All objects apart from the Heightmap (tessellated from heightmap PNG) are modeled by us with Blender and imported to Vulkan using TinyGLTF.
- Playable: The player can control the racecar and make his way around the track to set a new laptime.
- Advanced Gameplay: Additionally, going off-track triggers track limits and the lap is invalidated if the car is off-track for more than 0.1 seconds. This time limit gets reset every lap and is shown by a HUD element, visualizing the time off-track. Regarding the actual driving, the player has to shift through the gears and keep the RPM within an ideal range to extract the maximum amount of power from the engine. Shifting too early will result in slow acceleration while eager downshifting without lowering the speed accordingly can even lead to a total engine failure, forcing the player to retire the car.
- Min 60 FPS and Framerate Independence: The whole game was implemented to work framerate independent through delta time calculations and runs constantly above 60 FPS.
- Win/Lose Condition: As the game logic is complete, the player can set lap times and is punished for track-limits.
- Intuitive controls: Our game is primarily built to be played with a PS4 controller but can also be played using a keyboard.
- Intuitive Camera: The camera resembles a first-person view from the driver's cockpit and emulates similar racing games.
- Illumination model: The whole scene is illuminated by a directional light (sun). All objects are shaded accordingly and cast shadows onto themselves and all other objects. When looking directly at the sun, cinematic lens flares are visible.
- Textures: All objects are textured and resemble a realistic racetrack placed in a valley, surrounded by a hilly grass terrain.
- Moving Objects: The racecar moves freely around the track and can also bump into breaking boards, which are then thrown away and can be pushed around.
- Documentation: yes.
- Adjustable Parameters: A simple Menu has been implemented, enabling the player to toggle Fullscreen-Mode, Resume and Exit the game. The resolution for the windowed mode can be set before startup using "config.ini".

Optional:

- Collision Detection (Basic Physics): We use Bullet Physic's Collision Detection to trigger the track-limits and the completion of a lap. It also handles the collision of the car with the barriers around the track and prevents it from leaving the racetrack area.
- Advanced Physics: When colliding with the breaking boards, they emulate real foam breaking boards and are thrown into the air and can also be pushed around on the ground.

- View-Frustum Culling: By pressing [F8], it can be enabled and helps to boost the games performance. All objects except the heightmap, racetrack, racecar and terrain are then culled. By pressing [F4], a debug camera is enabled that can move with [I,J,K,L], go higher/lower with [U,O] and rotate using the left mouse button.

Effects:

Lighting:

- Shadow Map with PCF: All objects cast shadows onto themselves and all other objects by creating a shadow map every frame. Only the heightmap's shadow map is just computed on startup and then baked into the scene, to save computing power, as it stays perfectly stable. For all shadows, PCF is implemented to smooth the edges. Additionally, Cascaded Shadow Mapping is incorporated to dynamically render close shadows with more fidelity and those far away with less resolution, to save computing power. Shadow Acne is prevented through a depth bias, introduced in the Shadow Renderer, a dynamic normal offset in the shader and by enabling front-face culling for suited objects. Peter Panning is mitigated through tuning each of the bias values introduced in preventing shadow acne, dynamically enabling front-face culling only for solid objects and leveraging the higher shadow resolution for closer objects, introduced by Cascaded Shadow Mapping.

Terrain:

- Tessellation from Height Map: The mountain range surrounding the track is being created by tessellating a heightmap using a grayscale heightmap PNG file and corresponding shaders (Tessellation Control Shader, Tessellation Evaluation Shader). The Wireframe mode can be toggled using [F3] to better visualize the tessellation steps.

Animation:

- Hierarchical Animation: The racecar's tires and steering wheel move upon driver input and stay attached to the racecar.

Texturing:

- Environment Map: Environment Mapping is implemented using a cube map sampling approach to provide realistic reflections on the car's metallic surfaces. We generated custom environment textures in Photoshop, which are sampled within the fragment shader using the reflection vector calculated from the surface normal and view direction to get the final pixel color.

Post Processing:

- Lens Flares: Vulkan Occlusion Queries are used to determine the visibility of the 3D sun object. The sun's world position is then projected into 2D screen space to calculate a vector pointing toward the center of the camera view. Using a dedicated additive blending pipeline, the system then renders the primary sun glows alongside predefined ghost and halo textures to form a beautiful and cinematic lens flare. These artefacts are then scaled and spaced along the screen-center axis and fade in smoothly as the sun comes into sight.

Walk-through:

1. **Startup & Configuration:** Before launching the game, you can adjust the window resolution and fullscreen mode within the config.ini file. Upon starting, you are placed in a first-person cockpit view on the starting grid.
2. **Basic Controls:** The game is optimized for a PS4 controller but can also be played via keyboard (WASD to move, Up/Down Arrows to shift).
3. **Manual Gear Management:** The car utilizes a custom manual gearbox and engine logic. You must monitor the RPM gauge on the HUD and shift up as the engine reaches its peak to maintain acceleration.
4. **Track Limits:** Drive around the asphalt circuit to set a valid lap time, which is tracked on the right side of the HUD. If you drive onto the grass, an "OFF TRACK!" warning will appear. If your total time spent off track exceeds 1 second, your current lap will permanently be invalidated, triggering a red "LAP INVALIDATED" banner.
5. **Physical Interactions:** Navigating the track requires avoiding the solid metal barriers, which will physically block and stop the car. Conversely, the 50m and 100m foam braking boards are dynamic rigid bodies that you can freely smash into, sending them flying into the air and tumbling across the ground.
6. **Debugging & Testing Tools:** We have included several hotkeys for the grading process to easily inspect our technical implementations:
 - [ESC]: Opens the Pause Menu to toggle Fullscreen Mode, resume the game, or quit to the desktop.
 - [F3]: Toggles Wireframe mode on the terrain, allowing you to visually inspect the dynamic heightmap tessellation in real-time.
 - [F4]: Detaches the view from the car and enables the Debug Camera. You can move with [I, J, K, L], change elevation with [U, O], and look around using the Left Mouse Button.
 - [F8]: Toggles View-Frustum Culling on and off. You can check the console output to observe how many objects are actively being culled to save performance.
 - [R]: Manually resets the car and physics back to the starting origin if you get stuck.
 - [T]: Manually starts or stops the lap timer.