

193.019 Programming Techniques for Visual Computing

Submission Documentation

Jakub Profota (12549250)

Checklist

Gameplay

- 3D Geometry (6 points)
- Playable (3 points)
- Min. 60 FPS and Framerate Independence (3 points)
- Win/Lose Condition (3 points)
- Intuitive Controls (2 points)
- Intuitive Camera (2 points)
- Illumination Model (2 points)
- Textures (2 points)
- Moving Objects (2 points)
- Documentation (1 point)

Optional Gameplay

- Collision Detection (Basic Physics) (6 points)
- Advanced Physics (4 points)
- Heads-Up Display (4 points)

Effects

- Shadow Map with PCF (16 points)
- CPU Particle System (8 points)
- Tessellation from Height Map (12 points)
- Vertex Shader Animation (8 points)
- Triplanar Texturing (6 points)
- Contours via Edge Detection (12 points)

Total 102 points



Brumm-Brumm-Bau

Drive your trusty offroad boomer car through a pine forest in rocky hills. Race through checkpoints and try to get the best time! Yes I know, this should have been a road construction game, but I have no idea how roads are being constructed, okay? So chill out in the woods!

The offroader is controlled with arrow keys. You can rotate the camera while holding the left mouse button. Use scroll wheel to zoom in and out.

You can also toggle a debug ImGui interface with I key that shows some statistics like FPS and camera position. If you then press 0 (that is O like Orangutan, not zero), a debug view with various options and sliders will appear. Here, you can toggle between an orbiting and free camera. The free camera is controlled with WASDQE keys and holding left mouse button. If you press K, a debug render options window will appear, allowing you to switch between various debug renderers.

Used PTVC Framework and its dependencies + Jolt physics engine + cgltf glTF loader.

3D Geometry

There are three geometries loaded (star, tree, vehicle), see `assets/models` folder for licenses and links.

Playable

Well the game starts and you can drive around and collect checkpoints!

Min. 60 FPS and Framerate Independence

Render is unbounded, physics step is fixed to 60 FPS.

Win/Lose Condition

You can win by collecting all checkpoints and you *can* loose by falling of the edge or flipping the car.

Intuitive Controls

Arrow keys.

Intuitive Camera

Left mouse button.

Illumination Model

The best example of this is the vehicle. It has a texture, material properties, normals, and is lit by the directional light of the sun.

Textures

The ground is actually triplanar textured!

Moving Objects

The vehicle moves around (if you push it with the arrow keys).

Documentation

Hello there!

Collision Detection (Basic Physics)

You do not clip through the terrain and can crash into the static trees.

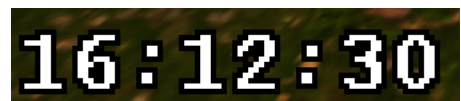
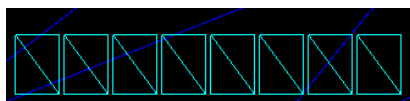
Advanced Physics

The car uses a 6 degrees of freedom Jolt physics vehicle model. It was kinda hard to tweak to get a satisfying driving experience that could climb the hills. However, the wheels get stuck into the ground on heavy impacts from time to time. Each wheel is attached to the body with a spring. You can see that when accelerating and braking, see the images to the right.



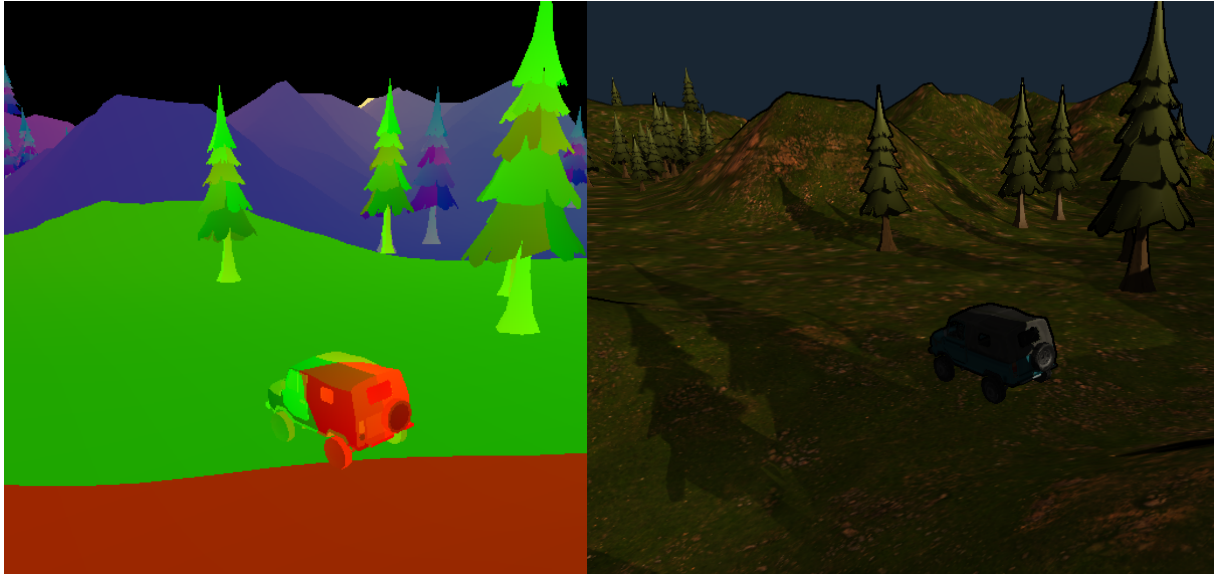
Heads-Up Display

Monospaced bitmap font, each character is a quad.



Shadow Map with PCF

The shadow map is cascaded, because the camera is always very close to the vehicle and the shadows looked pixelated otherwise. There are three cascades rendered to 6144×2048 pixels texture. The orthographic view frustum (from the sun POV) is split into sub-frustums and the scene gets rendered to the depth image. When rendering an object, a cascade is selected in the fragment shader based on the distance from the camera. The world position is transformed to the light space and the selected cascade is used to map to the cascade atlas texture. A bias is applied to compensate for slopes (larger world-space texels) and farther cascades. 2×2 PCF samples are read from the depth texture, compared against the light space position and averaged. Back faces are rendered in the shadow pass to prevent self-shadowing front faces. The visualization of the cascades can be seen in the UV debug render.



CPU Particle System

The car emits exhaust gas particles when accelerating (when up arrow key is pressed). They are rendered as small instanced cubes that slowly fade away (shrink). Each has a little drift so their positions vary a bit. They are spawned at the location of vehicle exhaust in world space, so they trail behind.

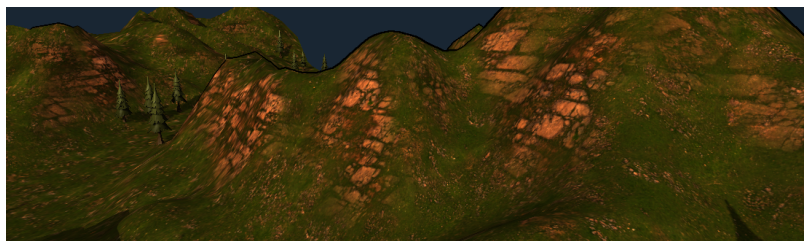


Vertex Shader Animation

The star is not rotated by a rotation matrix prior to feeding the data to the CPU, but the vertices are rotated by a matrix built in the vertex shader. The matrix depends on both elapsed time and the distance to the vehicle, with the star rotating faster as the car approaches the checkpoint. Kinda lame, but it is a vertex shader animation... :)

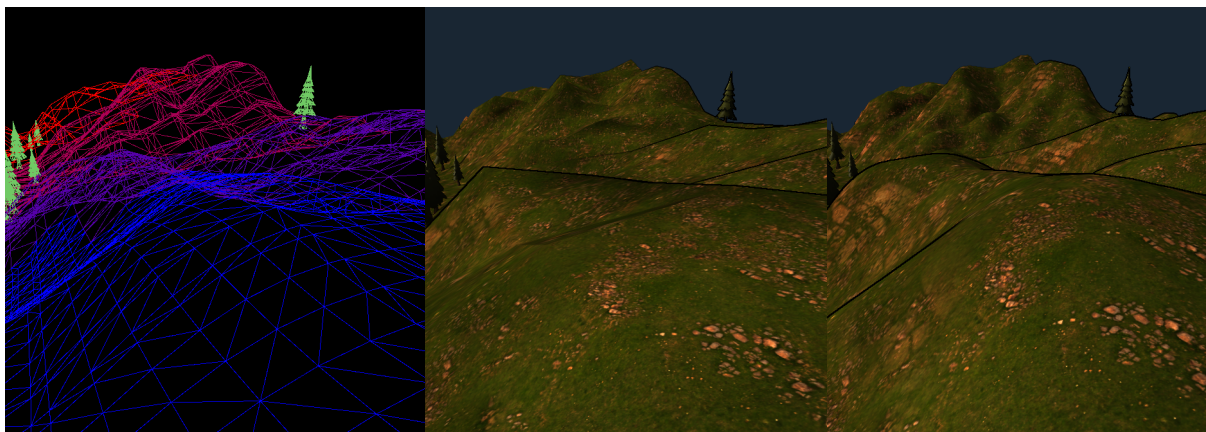
Triplanar Texturing

Each terrain texture (three of them called Rocky Terrain) is sampled and they are blended together based on the normal vector of the terrain.



Tessellation from Height Map

The collision shape and the rendered geometry are created from a grayscale heightmap, so you can load in your own heightmap and create your own level (btw, positions of checkpoints and trees are also loaded from an image, so you can *TOTALLY* create your own level!). The terrain is tessellated in shader, see the Wireframe debug render. You can tweak the tessellation factor, compare the images. The terrain's base mesh is 31x31 quads and can subdivide in four levels up to 8 subdivisions per quad edge, producing 64x64 small quads within the base quad. The new vertices are then displaced according to the heightmap and normals are computed from the neighboring heightmap values.



Contours via Edge Detection

3x3 Sobel filter applied on the depth buffer, a screen-space post-processing effect. It is drawn by overlaying the whole game window with a huge triangle and drawing black contours if a sharp change in the depth buffer is detected. The threshold of identifying a depth change as sharp is tweakable in the debug view, but do not overdo it, see image. You can also switch to Depth debug render.

