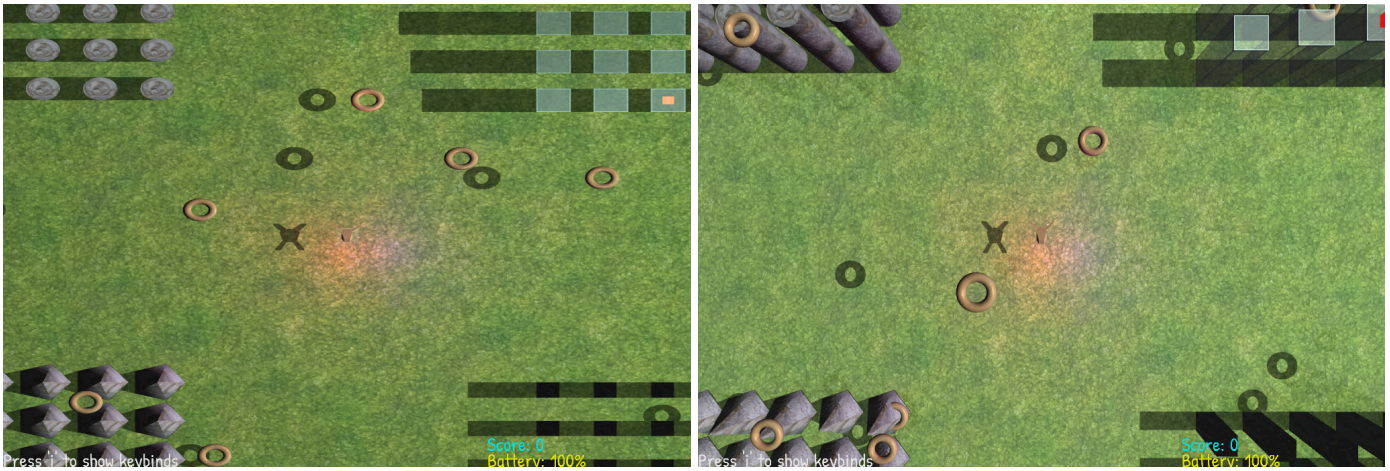
Relatório AVT 2025/2026

Grupo 13
99313 - Ramiro Moldes
103617 - João Santos
107095 - David Quintino

Trailer URL: https://drive.tecnico.ulisboa.pt/download/851498741624655

Cameras

We have 3 cameras implemented: a perspective and a top camera, both in a top view, and a camera that follows the drone and can orbit around it by left-clicking and dragging the mouse
They are created and put into a vector called cams and are updated by user keyboard presses.
(Orthographic on the left, and projection on the right.)

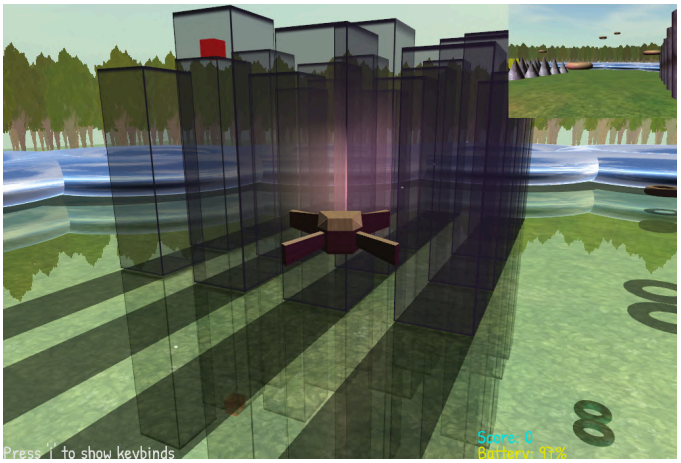


Lighting

We have implemented point, directional and spotlights. There are 3 point lights just hanging in the middle of the scene, The directional light is above the scene to simulate sunlight and the spotlights are used as headlights for the drone. Each of them are able to be turned on and off.
(Scene with directional [sun] light turned off, showcasing point and spotlights)

Transparency

Transparency is implemented in two different ways: one shown in the background, with billboard trees, square quads that discard fragments; and the other is with alpha blending, requiring a lot more care, such as disabling face culling and requiring sorted draw calls.

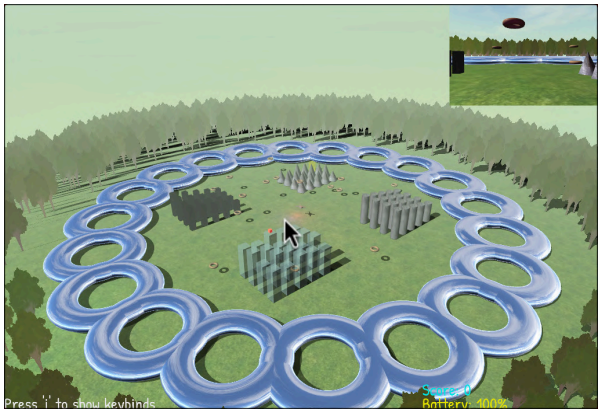


Collision detection

In order to implement collision we had to add collision boxes to every object that we wanted to have them. We used AABB boxes and implemented a collider to create the boxes, add them to the objects and detect collisions. Finally, in the collidable objects we implemented a function to decide what to do in case of collision. For the drone, he loses all speed and a bit of battery, the buildings and floating objects just serve as obstacles and the package uses them to know when it is picked up and dropped off.
Also we added a debug mode where you can see the hitboxes

Fog Effect

The fog uses a squared distance exponential for normal objects, however it is also applied to the skybox for a more consistent look where it uses the height as distance so the horizon is the most foggy and straight up has no fog, also there is a compensation factor for the edges as it is a cube instead of a sphere.



Rearview Camera

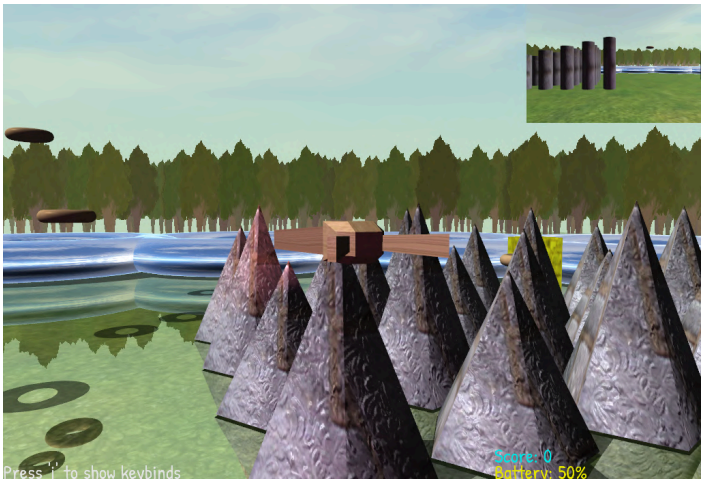
The rearview camera is achieved by drawing it before anything else as an orthographic projection into a quad on the top right corner. Stenciling is used to restrict the drawn content.

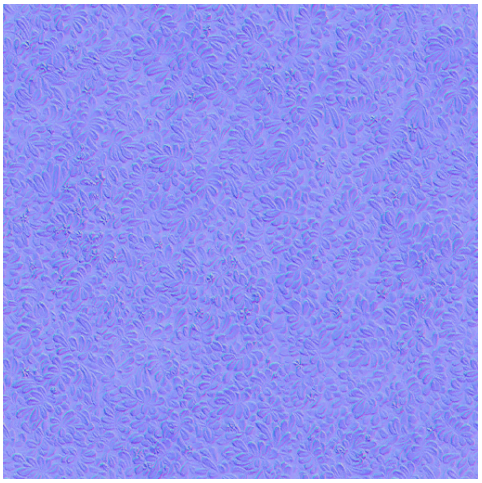
Billboard Behavior

The trees that surround the city are flat quads that display a texture, these quads are rotated so the XZ components of the normal always face the camera, the Y component is kept so rotation is fixed vertically. The shadows in the example show how the trees all have different rotations.

Bump Mapping

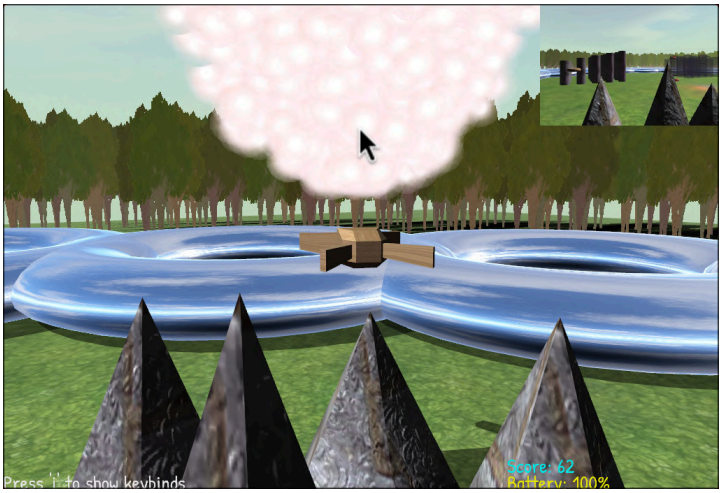
Bump mapping is applied to the cones and cylinders that have the stone texture. In the vertex shader, they input the normal and the tangent vectors, where the Gram-Schmidt is applied and the bitangent is calculated to complete the TBN matrix. The matrix is passed onto the fragment shader where it is multiplied with the normal that is read from a normal map, an external texture, and this is the normal that is used instead of the vertex normals.





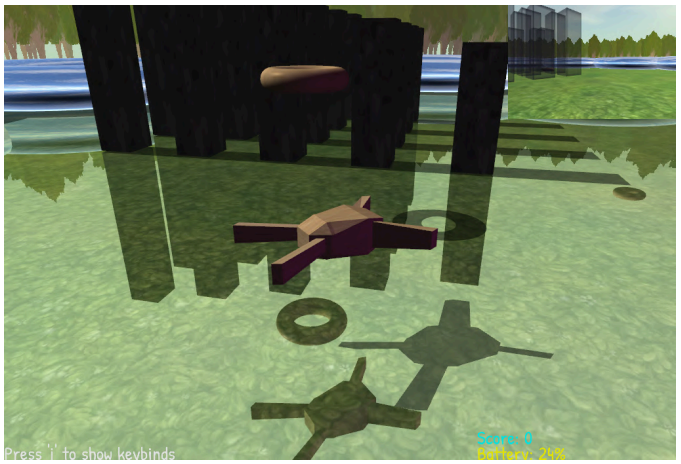
Particle system

When the player completes a delivery the particle system creates a firework explosion. Each explosion is made of a lot of tiny particles that get a random horizontal direction and a bit of downwards acceleration to add gravity.



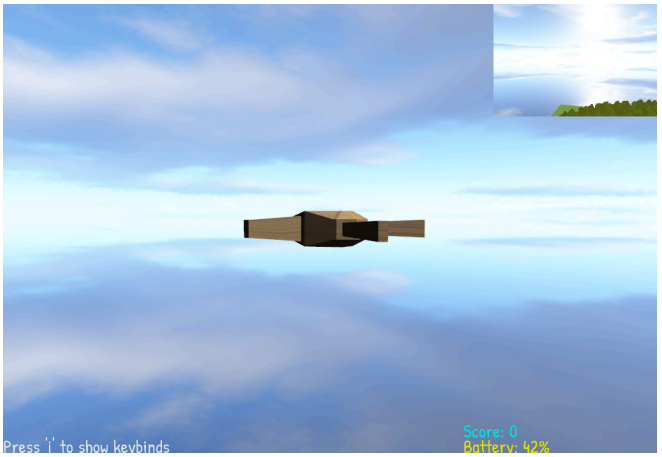
Planar Shadows and Planar Reflections

The planar shadows are achieved by transforming the objects with a projection matrix that only takes the directional light into account, the objects are also shaded with a flat gray color instead of their textures. The planar reflections are achieved by drawing every object and light with a mirrored Y position and a mirrored Y scale. Both effects are restricted by the stencil buffer to only draw where the floor will be and then drawing the floor with a blended alpha so it looks like a reflection.



Skybox

The skybox is a unit cube with cull faces disabled whose position is translated to the camera's, it uses a special shader that directly maps the (view transformed) fragment position to a cube texture coordinates. The Z coordinate is set to 1 so it always renders behind every other object in the scene.

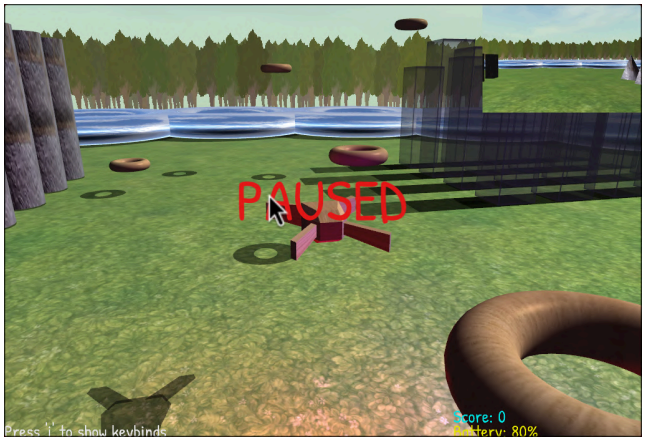


Environment Cube Mapping

This is set as a texture, but it is a special case that uses the reflected vector of the view direction and the normal vector to map to the skybox's cube texture coordinates. It is applied to the torus that surround the city.

HUD

The HUD is rendered using bitmap fonts in screen space after all 3D objects. It displays information on the battery level and score. Text is drawn with alpha blending and positioned using orthogonal projection for placement. The HUD is updated every frame and does not interfere with the 3D scene rendering. Pressing 'P' pauses the game and stops scene objects movement, and a restart screen appears on screen when the drone's battery reaches zero, on this screen pressing 'R' restarts the game.



Lens Flare

Lens flare is rendered as a 2D effect on top of the scene, simulating light scattering when the sun is visible from the camera. The flare position is calculated by projecting the light source into screen coordinates. Multiple textured quads are drawn with alpha blending to create the flare effect. Flare appearance and behavior are configured using the flare.txt. This is updated every frame and does not interfere with the 3D scene rendering.