Graphics Debug

Debugging the graphic performance of a level is vital to understand which parts are causing the bottleneck. It can also be used to find bugged/incorrect meshes or entities.

Start Depth.exe in Dev mode and then press F1 to bring up the Debug Toolbar. The two groups that are important are the “Debug texts” and “Graphics Debug”.

Debug texts

The important checkboxes here are Show FPS, Show Memory Usage and Show Render Info. These will tell you how costly the rendering of the scene is.

Show FPS

Shows how fast the level is running.

- **FrameTime**: Shows how long it takes to render a frame, this is the most important info.
- **FPS**: 1000 / FrameTime. It is what you usually see when talking about performance. It is easier to grasp but not as useful as FrameTime.

Show Memory Usage

- **Renderer Memory Usage**: The amount of VRAM used by the renderer. This contains frame buffers, shadows, terrain and post-effects. It is greatly affected by the resolution of the window.
- **Texture Memory Usage**: The texture memory used by the level. This contains all the textures used for the materials of the objects. Texture memory is reduced by a lot with just a few changes in the config file.
- **Vertex Memory Usage**: The amount of VRAM used by meshes and decals. The imporant difference between Vertex and Texture memory is that Vertex memory can't be reduced by changing a setting. It takes up as much space on a good and a bad computer.
- **Total Memory Usage**: The sum of the above memory usage.

Show Rendering Info

- **Draw Calls**: The number of objects rendered in the current scene.
- **Rendered Triangles / Vertices**: The total number of triangles and vertices rendered per frame.
- **Queries**: A special draw call used to check if an object is occluded. It is more expensive than a draw call.

Graphics Debug

To be able to find which parts of the scene that takes up the most performance you can use these special render modes.

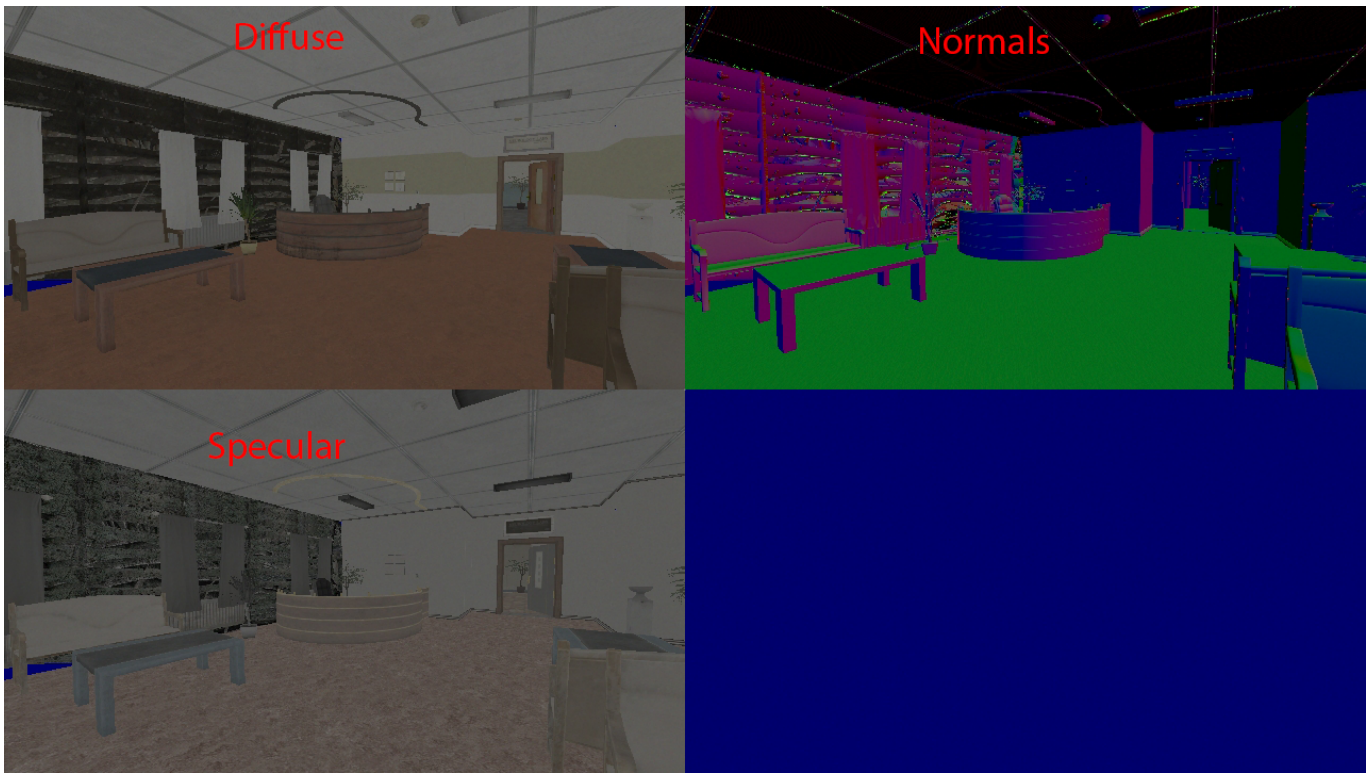
Previous Frame Occlusion

Pauses the occlusion culling and renders the scene with the occlusion culling from before the checkbox was ticked. This is used to see how effective the occlusion culling is in the specific scene. After activating this you can press F7 to fly around and look at the scene from another angle to see where the culling fails. Activating this will disable rendering of translucent objects since they can be unstable.

In the image to the left you can see what the player can see. It looks like only a few walls, some lockers and a door is rendered. On the right the camera has been moved back and it is now possible to see all the objects that get rendered this frame. There are a lot more than what it looks like in the left picture.

GBuffer

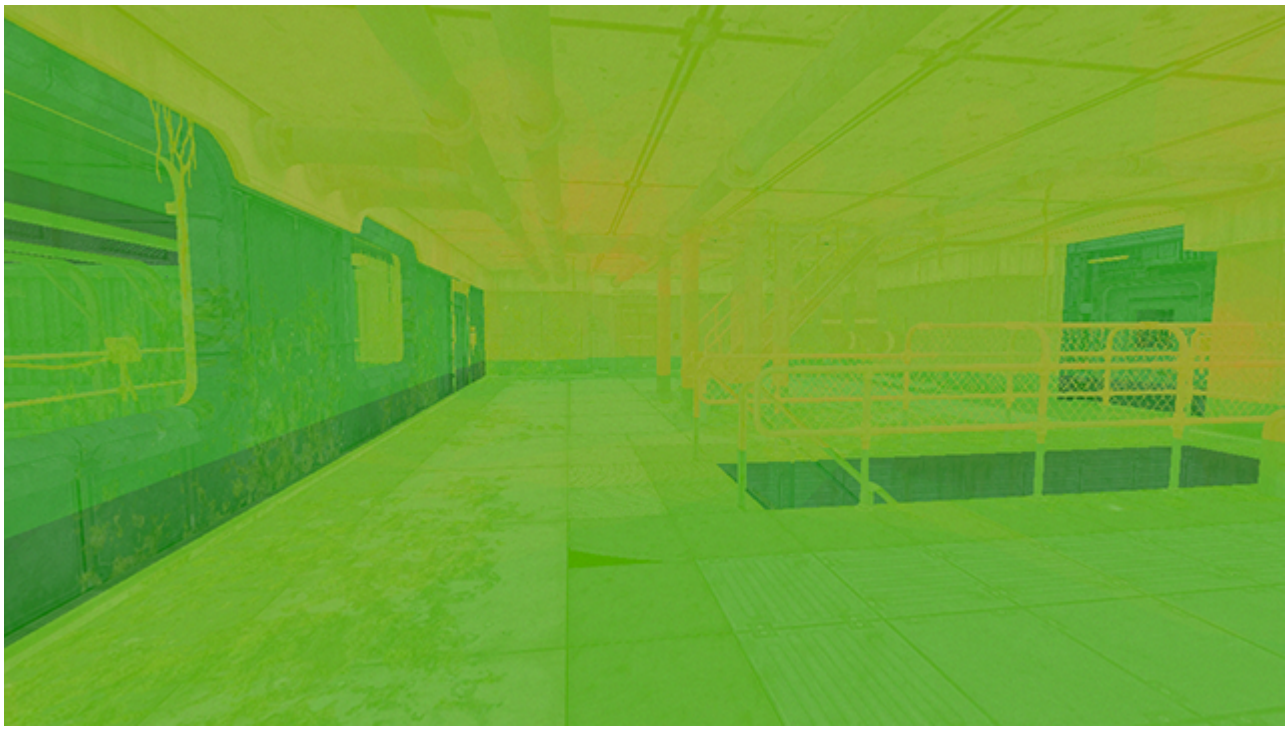
Shows what the frame buffers look like. These are then used by the lights to shade the scene. It can be used to check if a model has incorrect textures or normals.



Light Complexity

Light complexity shows how expensive the lighting of a scene is. It creates a heatmap on the screen to show how much power each light takes. Overdraw of light is not very expensive, it is about twice as expensive as a translucent object. When the light has a shadow map it becomes much more expensive. The complexity of a shadow casting light is based on how many draw calls and triangles are required to generate the shadow map.

The red areas on the right image is not good. There seem to be multiple large (in screen space) shadow casting lights in this scene.



To test if most of the complexity comes from shadows you can toggle the “Draw Shadows” checkbox on the Debug Menu. Here is what the scene complexity looks like without shadows.

Overdraw

Creates a heatmap that shows how much translucent overdraw each pixel receives. Overdraw is how many times a pixel gets rendered to. The goal should be that no part of the screen is red. It does not matter that much if only a small part is.

- Green: 0-16
- Yellow: 16-32
- Red: 32-48+

On the left you can see the scene rendered normally. The fog particles are barely visible here. On the right the pixel overdraw is rendered. The red areas show that more than 32 translucent planes are rendered, which is not good.