Entity 1

Position X: 5.0 Y: 0 Z: 0

Rotation X: 0 Y: 0 Z: 0

Scale X: 1.0 Y: 1.0 Z: 1.0

Reset Transform

Custom Shader

Edit Vertex
Edit Fragment
Reset Shaders

Change Model

Select a model...
Load

Texture

Default Diffuse

Default Specular

Load

Light

Point Light

Commands

Despawn

Scene Editor in OpenGL

A basic scene editor inspired by the likes of Unity

Amandus Søve Thorsrud

The Idea

- Free camera that can move around the scene
- Spawn & select objects
- Manipulate some of the objects’ properties (transform, texture, …)
- In-game shader editor

Quick Overview of the Implementation

- Implemented in Rust, with OpenGL 4.1 for platform compatibility
- Entity Component System using `bevy_ecs`
- Immediate mode GUI using `egui`
- Deferred rendering pipeline
- Shadow mapping

Entity Component System

Using `bevy_ecs`

- Needed a way to represent the world
- ECS is the hot new-ish thing and works well with Rust
- `bevy_ecs` from the Bevy game engine
- Entities are just identifiers corresponding to collections of components
- Systems operate on these components

UI

Using `egui`

- Immediate mode GUI library for Rust
- Designed to be easy to integrate into existing applications
- Implemented as a *system*
- Makes it easy to connect UI actions to component changes

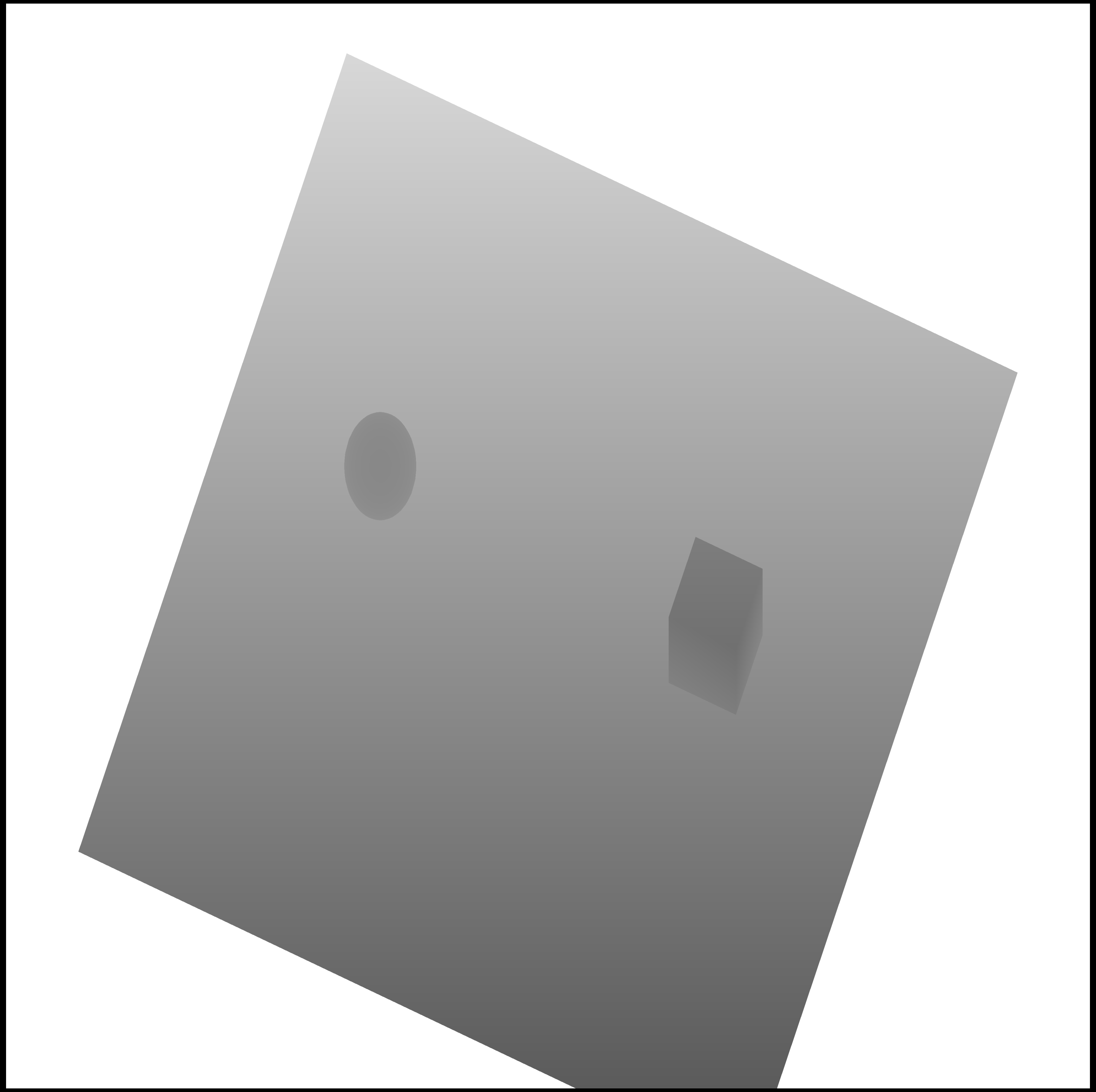
Renderer

- Also implemented as a *system*
- Looks for entities with `Mesh` components (as well as a few others)
- Draws in three passes:
  - Depth pass (shadow mapping)
  - Geometry pass
  - Deferred lighting pass

Shadow mapping

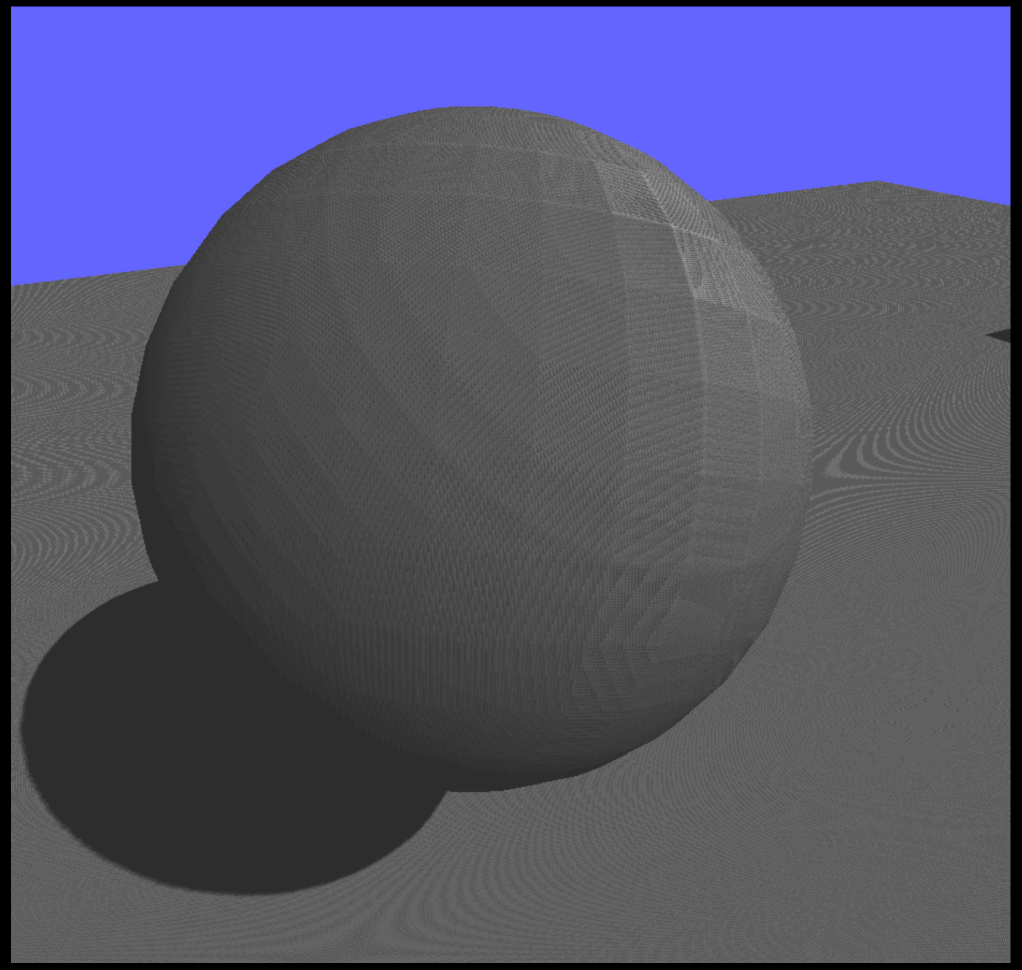
- Basic idea:
  - Render from the light’s perspective to frame buffer
  - Use depth information in final shading to draw shadows
- Use directional light as source for shadows, render from position along the lines of the light direction

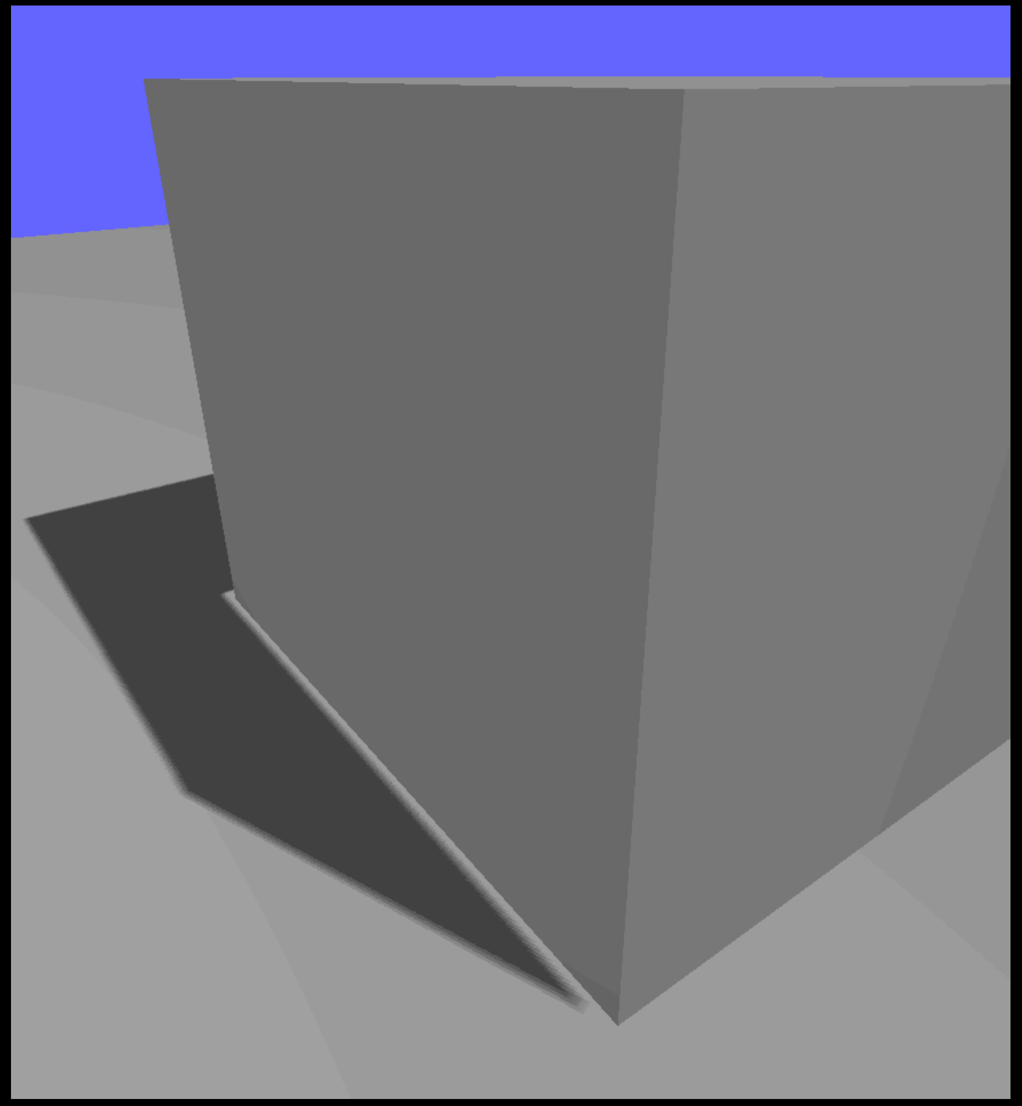
Visualization of Depth Pass



Problems with Shadow Mapping

- Limited area
  - Orthographic projection
  - Texture size
- Artifacts
  - Shadow acne
  - Peter panning





Improving Shadows

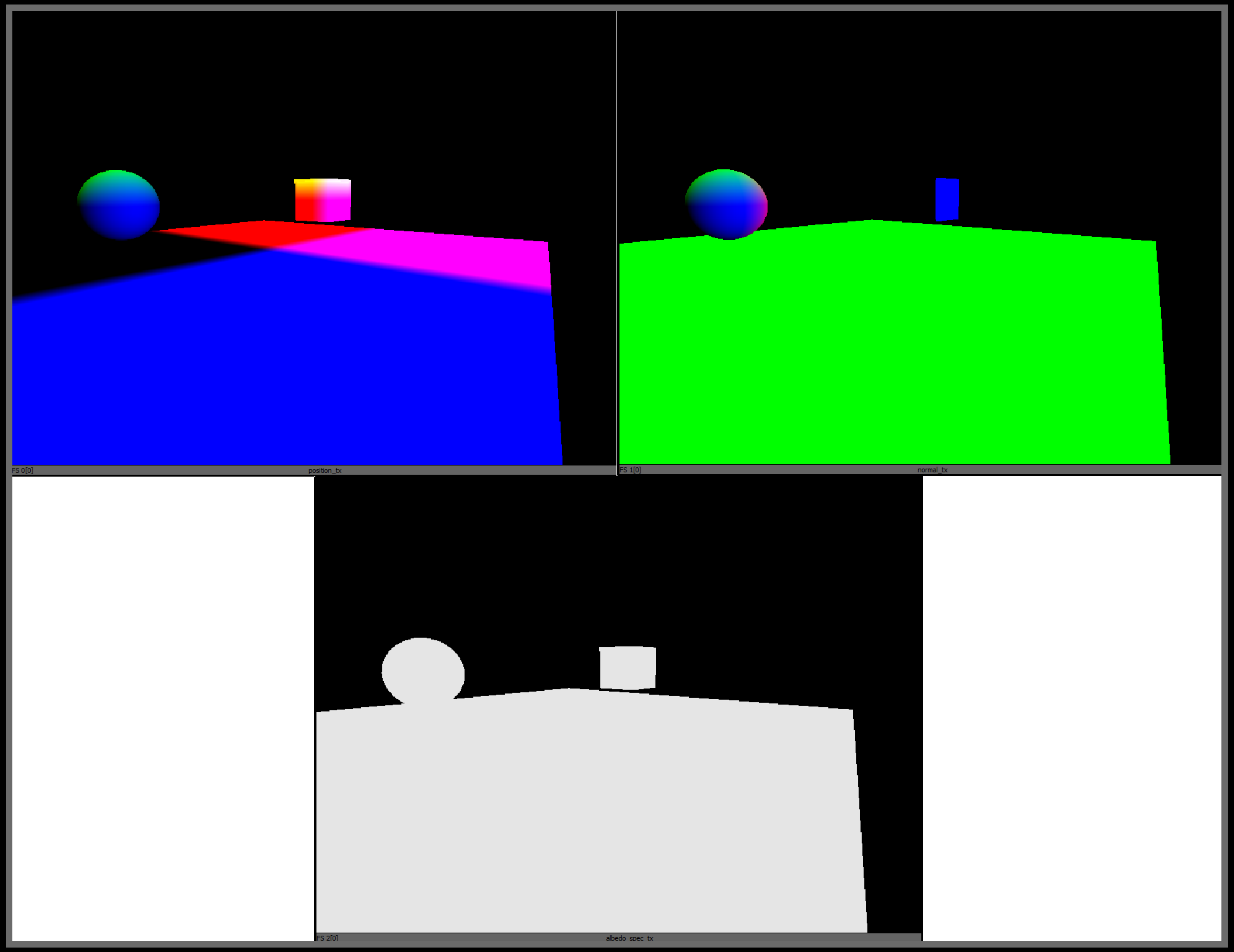
- Percentage-closer filtering
- Depth bias
- Front-culling
- Tight near/far planes of projection

Geometry & Deferred Lighting Pass

- Geometry pass:
    - Render to a frame buffer
    - Write per-fragment lighting variables to textures (position, normals, etc.), referred to as G-buffer
    - Write entity ID to stencil buffer for selection
- Deferred lighting pass:
    - Draw a quad covering the screen
    - Sample the above textures to calculate lighting

Visualization of the G-buffer

- Position
- Normals
- Albedo & Specular



Deferred rendering: Pros & Cons

- Pros
  - Large amount of point lights without heavy performance loss
    - Even more with light volumes (not implemented)
  - Process lighting for fragments **once**
- Cons
  - More complex pipeline
  - Increased memory usage
  - No blending (need to combine with forward rendering)

Demo!