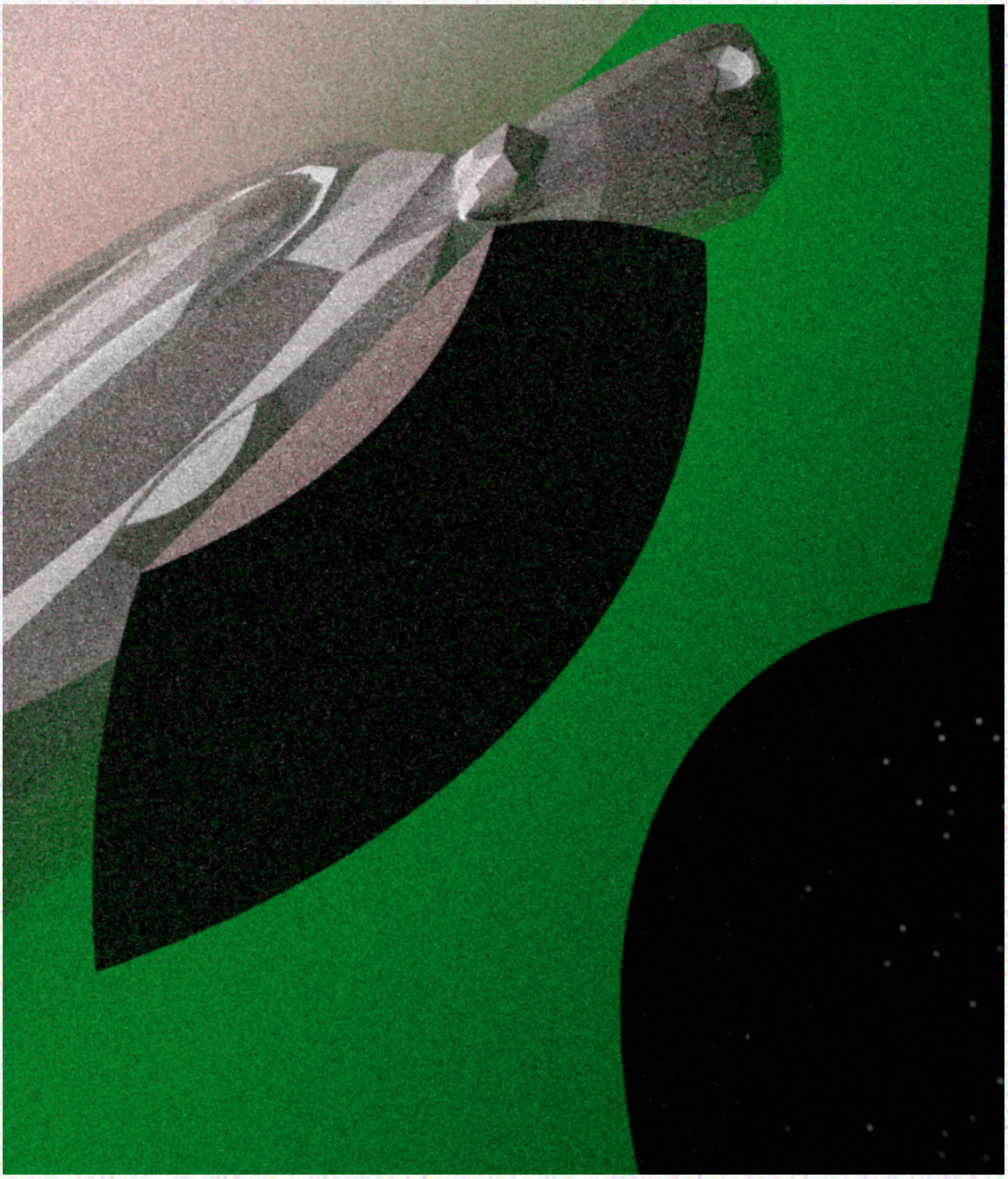
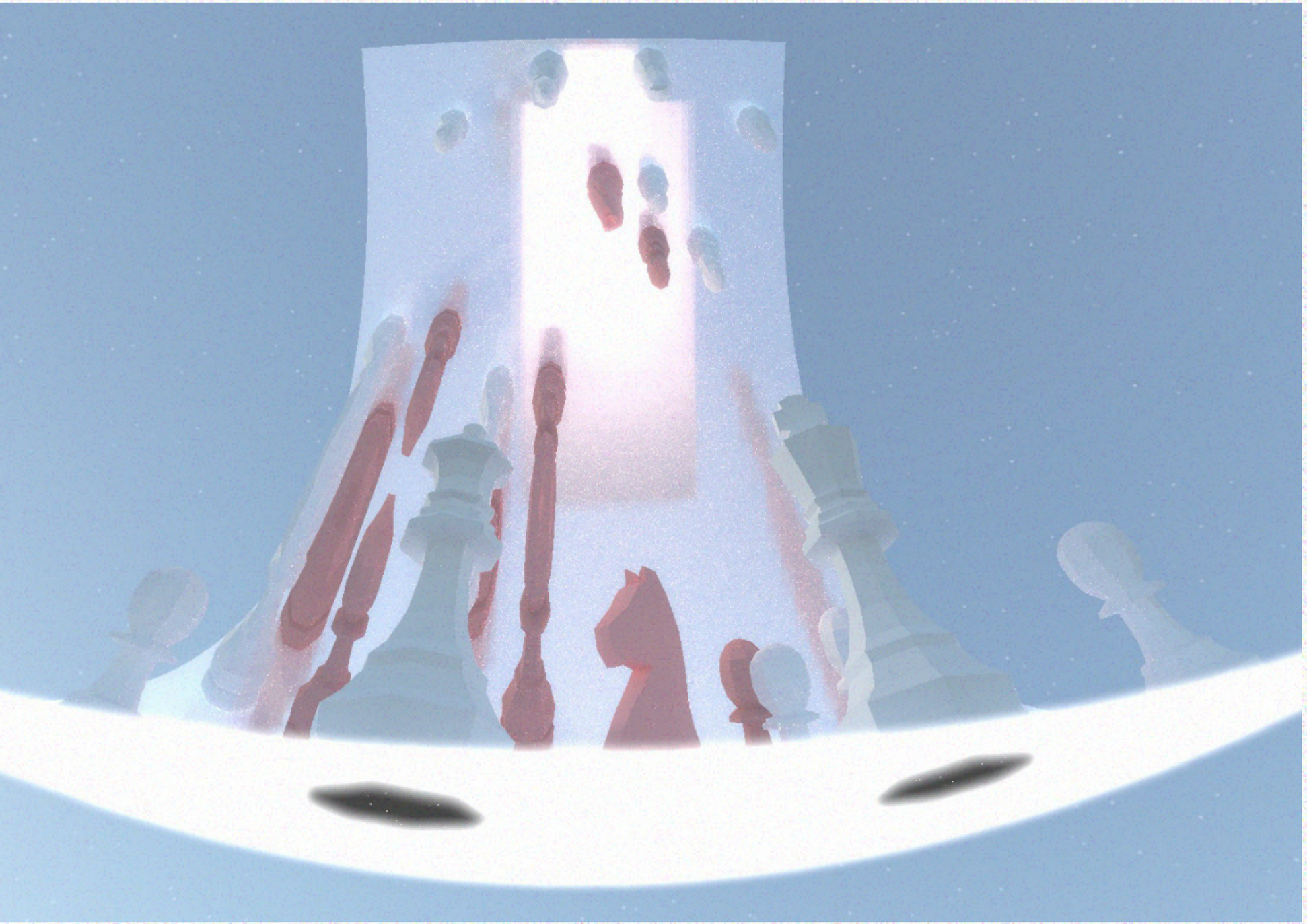
PORTFOLIO

ANDO DU



ancient.prophet@icloud.com https://ando-du.com

TECHNICAL ART



RAY-TRACING WITH A "TWIST"

HOW DO WE SEE? WHEN LIGHT IS RAIN — COMPUTE SHADER & DATA STRUCTURE

Powered by ReSTIR accelerated raytracing with a twist, this project showcases the overlooked potential of state-of-the-art technology for artist expressions in a game environment.

It starts with my reflection on why games are becoming increasingly photorealistic. Ray tracing is just one of the many emerging tools that make simple yet naive assumptions about the artistic style for games in the future. Ray tracing heavily relies on ray tracing hardware that is based on the same preconception. Other art styles are potentially in jeopardy because of the path dependence it might create. Imagine a future where photorealistic games can be highly optimised while others are slow and painful to run and play.

But, raytracing does NOT have to be photorealistic. It can be surrealistic as we show in this project. When "gravity" is programmed to bend light, we achieve an art style resembling the film Inception. My implementation is a modification of an open-source Unity path tracing package so with a little more work, it can also deal with complex texture, complex terrain, and almost everything one would need to render a real game.

Work done:
An updated Bounding Volume Hierarchy (BVH) data structure to accelerate our raytracing with a twist.
A new fast curved ray intersection algorithm to check whether a curved ray hits a BVH/a mesh bounding box (AABB)/a mesh triangle.
An updated illumination algorithm

Programming language:
C#, HLSL

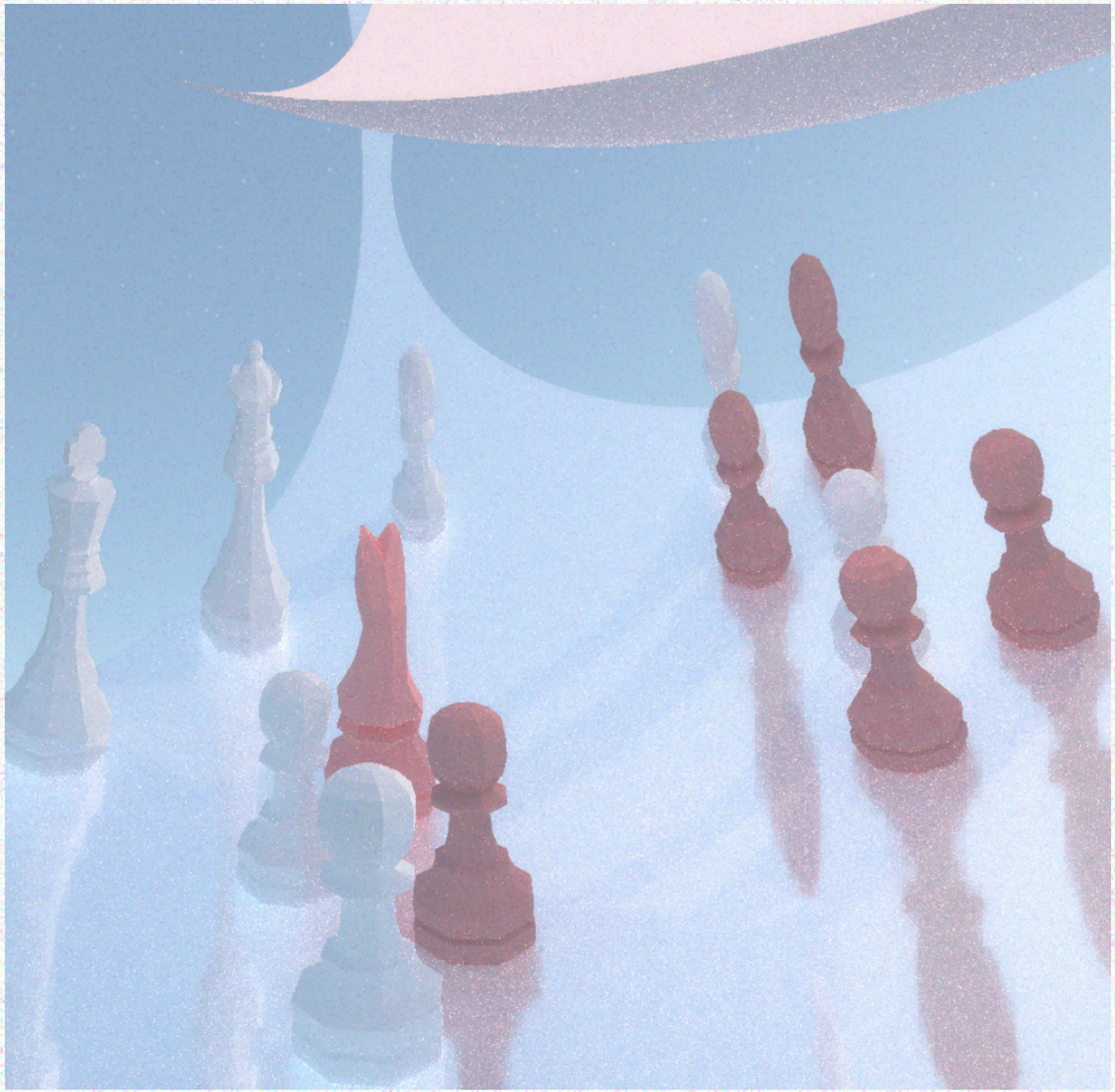
Special thanks to:
Pjbomb2 @ Github for their TrueTrace-Unity-Pathtracer package
H. Ylitie et al. @ Nvidia for their paper Efficient Incoherent Ray Traversal on GPUs Through Compressed Wide BVHs
Sebastian Lague @ Github for test scenes

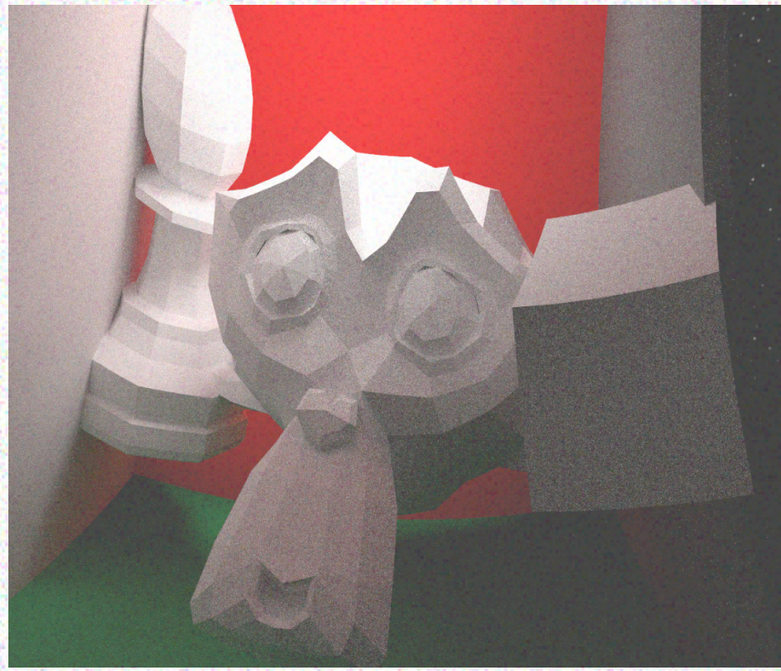
LEFT: chess scene rendering

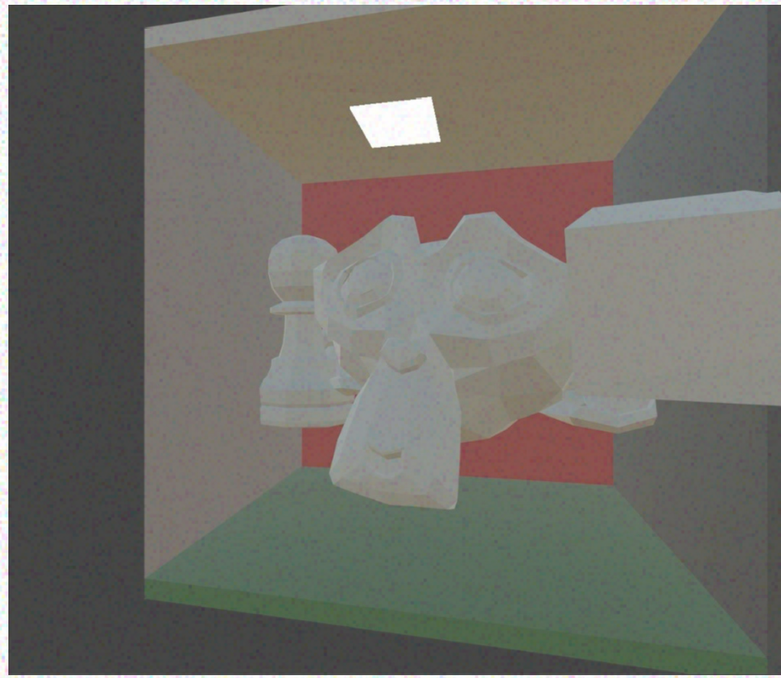
TOP RIGHT: rendering

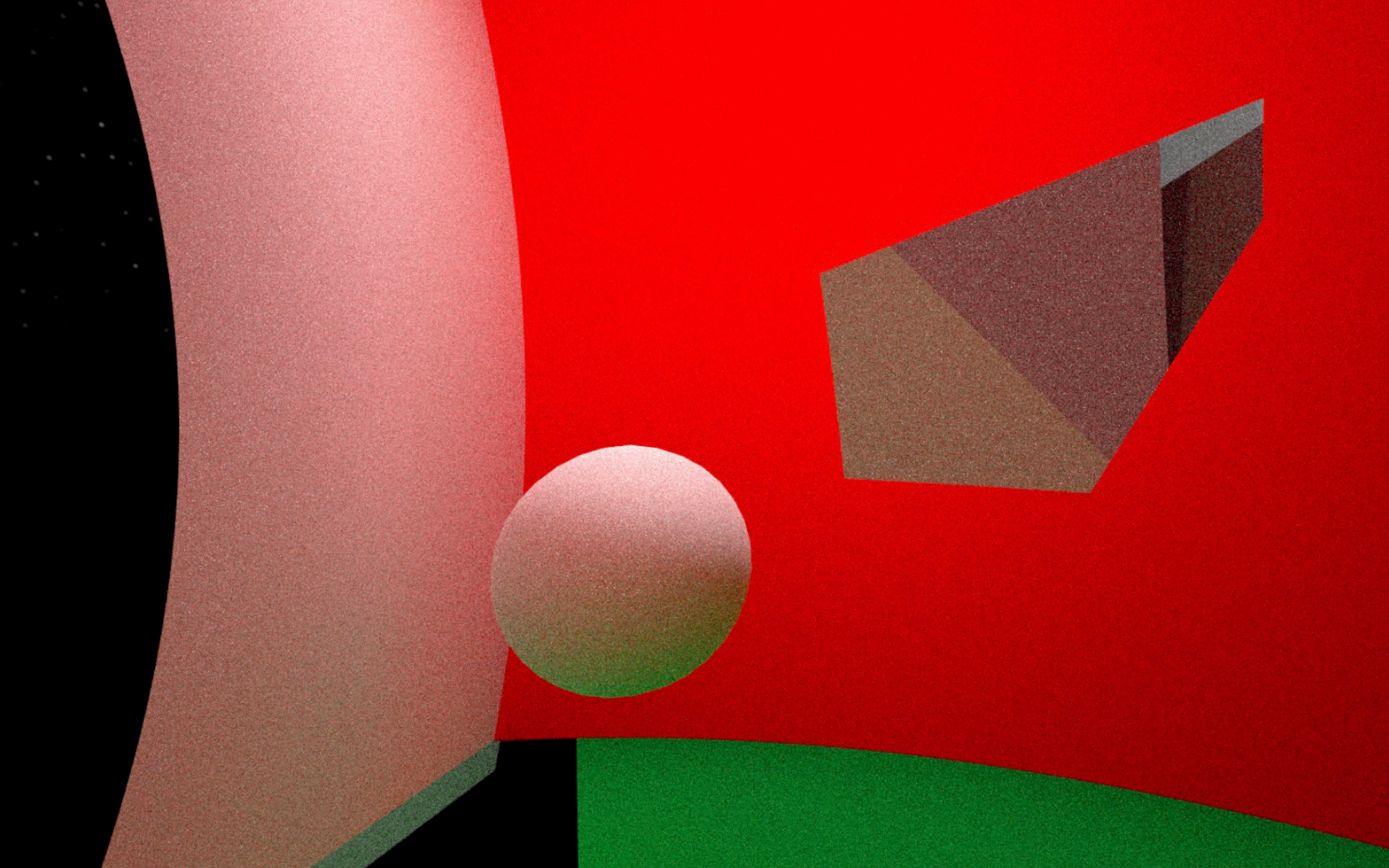
BOTTOM RIGHT: chess scene set

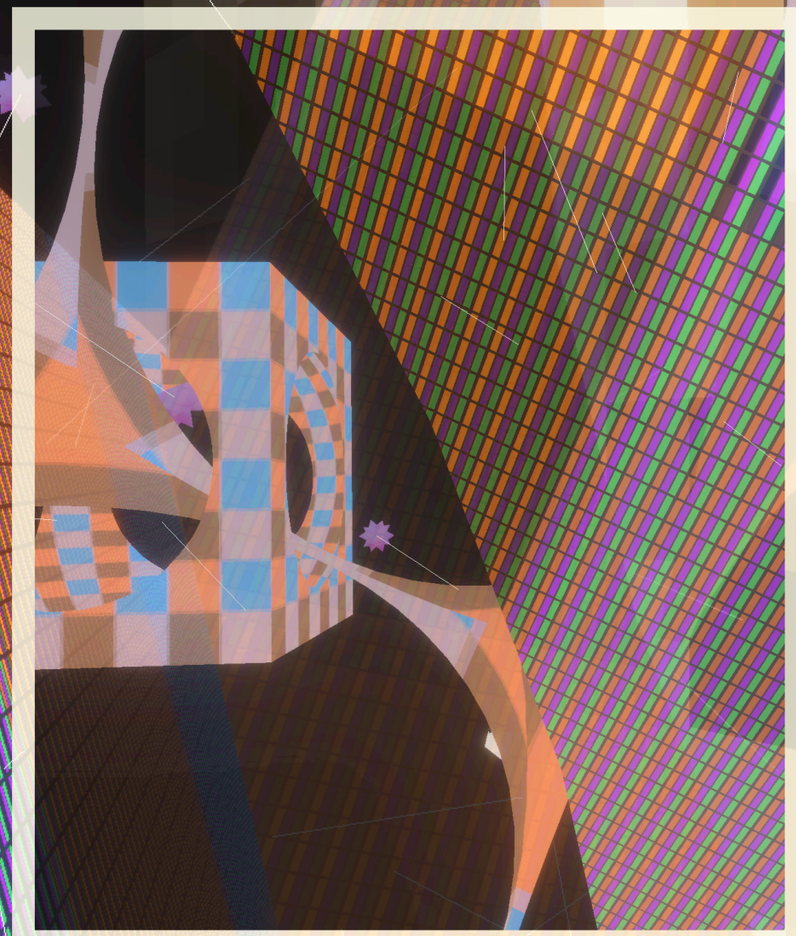
VIDEO on next page (might take some time to load)











LCD SCREEN COLOUR DISPLAY DEMO

CAN YOU FIND THE COLOUR YOU CANNOT SEE? – UNLIT SHADER

A game prototype that asks questions about our colour perception. Have you ever wondered what is behind the digital display on your phone? Are you seeing the true colours? Or, is it an illusion?

This project offers a rare opportunity to see through the facade and reflect on how colour is perceived in our digital world. You might know the RGB representation of colours on a computer. So, what about a CMYK representation, or can there be even more freedom? Well, you will see when you find the hidden colour!

Work done:
Smooth game control and navigation
Modelling and texture mapping.
Colour deconstruction algorithm (HLSL)
A low-tech solution to the one-way flow of information from C# script to HLSL shader for game progression and gameplay feedback

Programming language:
C#
HLSL

Special thanks to:
Sion Fletcher for Technical Art Support

DIGITAL COLOUR BLENDING & DEMO WEBAR PORTING

CAN YOU FIND THE COLOUR YOU CANNOT SEE? NO.2 — CUSTOM RENDER PIPELINE

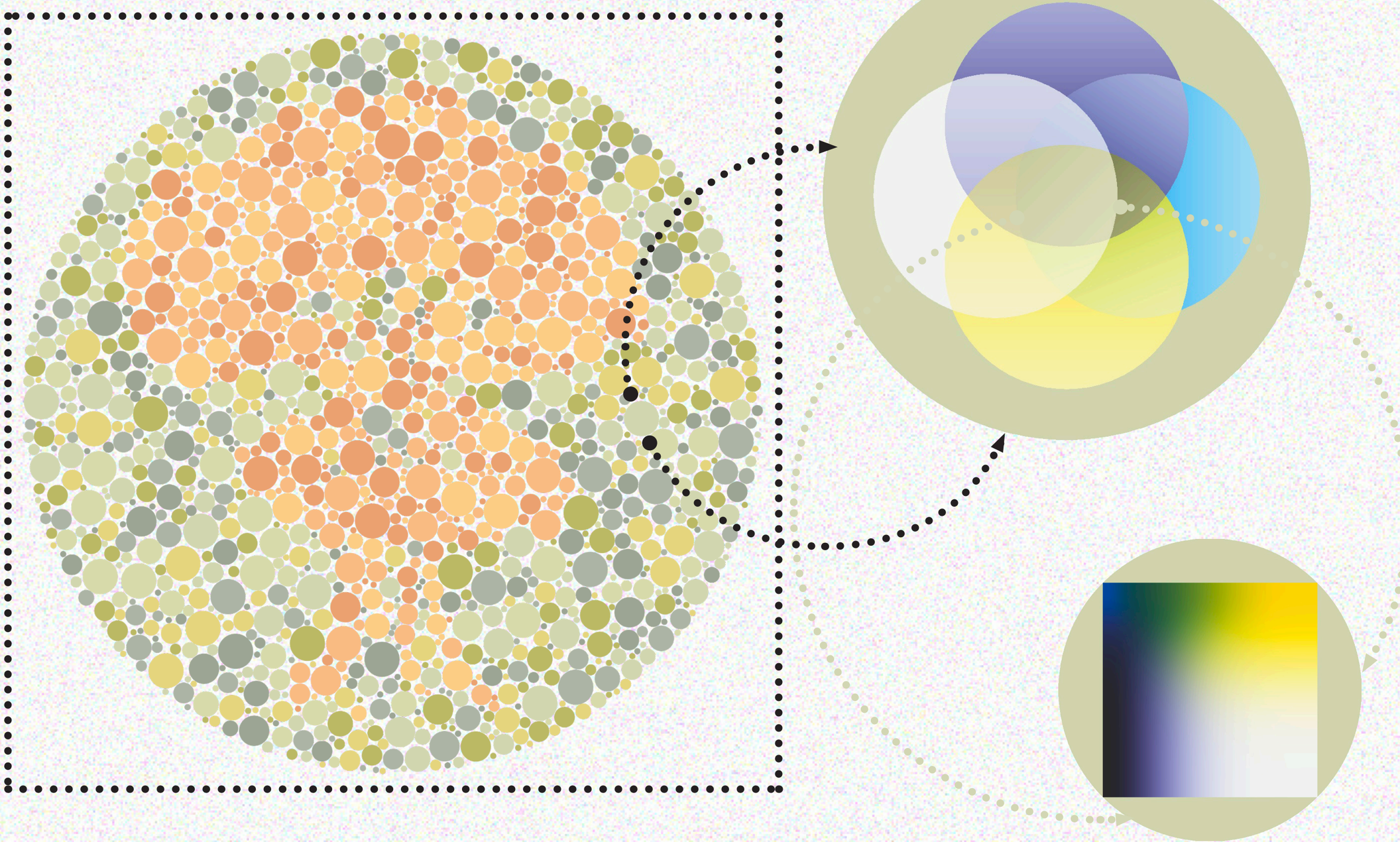
A game prototype that asks questions about our colour perception. Have you ever wondered what is in the oil paint? Are you seeing the true colours? Or, is it an illusion?

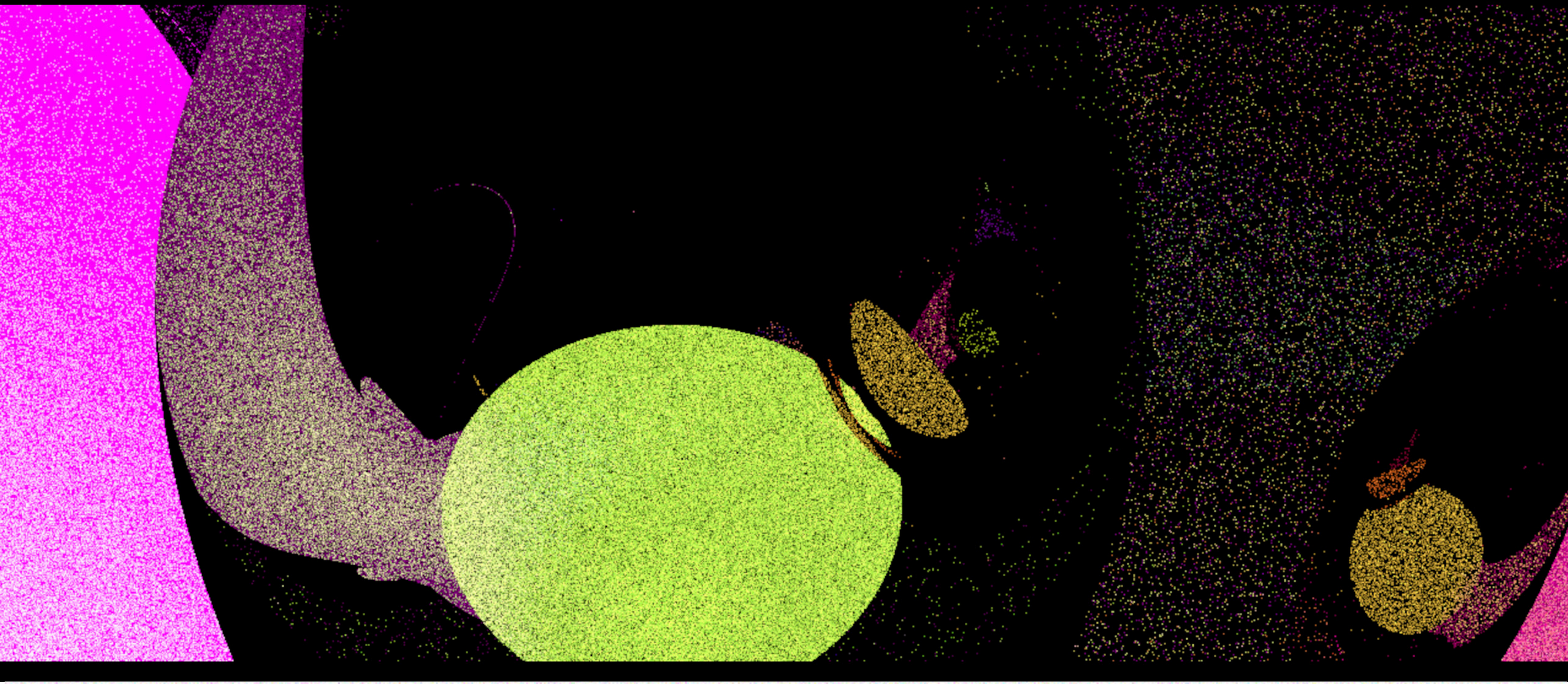
This project offers a rare opportunity to see through the facade and reflect on how colour is perceived in both our digital and physical world. You might know the RGB representation of colours on a computer. But what makes oil paint so distinctively different?

Work done:
Unity WebAR space and speed optimisation
Render Queue order hack
Digital oil mixing algorithm (R & HLSL)
Smooth (40fps+) Unity AR experience requiring only a web browser

Programming language:
C#
HLSL
R

Special thanks to:
Sion Fletcher for Technical Art Support

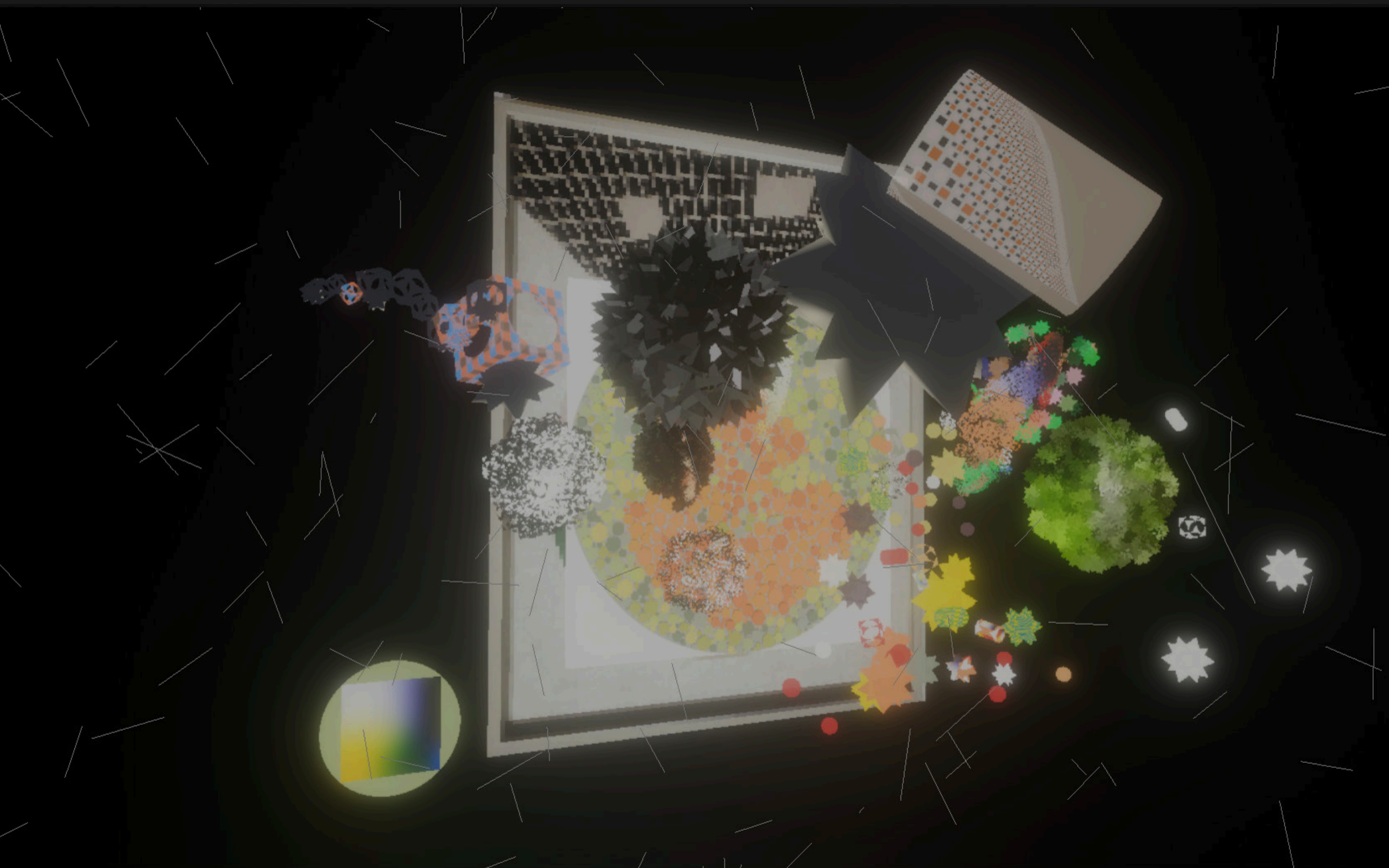




RAYTRACING WITH A “TWIST” NO.2 DEMO

COMPUTE SHADER & CAMERA PATH ANIMATION

Special thanks to:
Sebastion Lague @ Github for models
Sion Fletcher for technical art support



CPU SIMULATED 3D-PIXEL VIDEO PLAYER DEMO

NUCLEAR AFTER LASZL6 MOHOLY-NAGY – CPU SIMULATED GPU RAYTRACING & COMPUTE SHADER

A showcase of my understands of compute shader and unlit shader and how to opitimise them when simulated using CPU. Pixels are not merely 2D pixels but also can be 3D shapes including Platonic solid, stars, and spheres.

Work done:
WebAR porting
Real-time in-editor scene buidling tool
CPU simulated Raytracing
CPU simulated Compute Shader
CPU simulated Unlit Shader
3D models as "2D pixels"

Programming language:
C#
HLSL

Special thanks to:
Sion Fletcher for Technical Art Support