

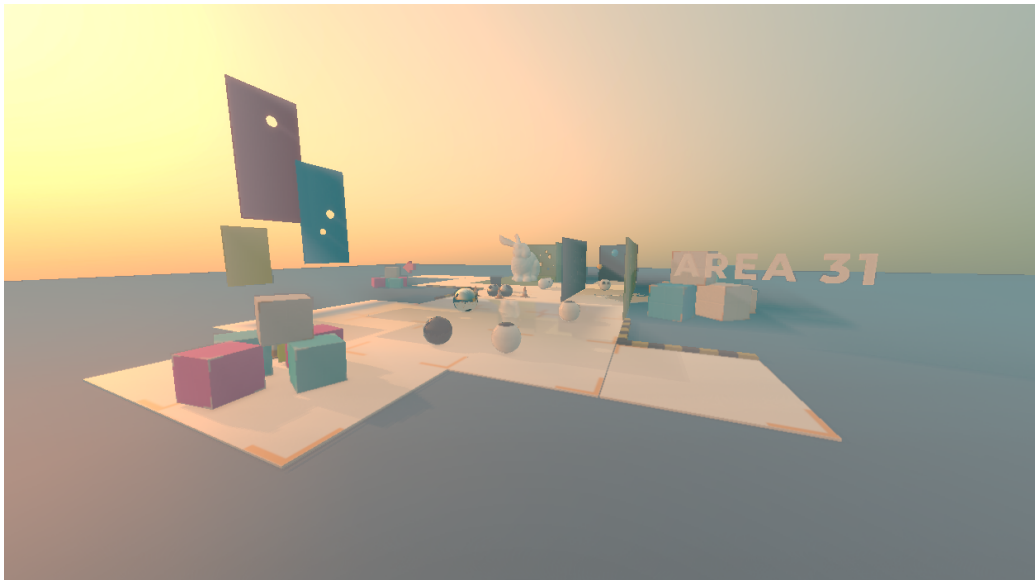
# EZG 18 :: Area 31 :: Submission 2

Group 31

Kurt Christian Chabek, 09901743

Lukas Herzberger, 01006039

January 21, 2019



## Controls

- **R** to restart the animation
- **C** to switch to a debug controller: move around with **WASD** and **mouse**
- **Space** to pause the animation
- **Q** or **ESC** to quit

## Effects

The scene builds on the engine developed in SS 2018 for the CG Lab course, which already implements PBR-shaders, normal mapping, and a physics engine.

New effects implemented for this course are:

- Deferred lighting
- Image based lighting
- Dynamic cubemaps, parallax corrected reflections
- SSDO
- Light shafts
- Bloom
- HDR and Tonemapping

### Deferred lighting

Deferred lighting happens in two stages: first the G-Buffer gets filled, lighting gets applied second. The G-Buffer consists of four render targets: position, normals, albedo, and one render target for metallic (r-channel) and roughness (g-channel).

### Dynamic cubemaps, parallax corrected reflections

A reflection probe is essentially a dynamic cubemap with a defined bounding box. Every object inside this bounding box uses the probe's cubemap as reflection and lighting source. (You can see the soft edges of the probes in the scene!)

Cubemaps represent an environment at infinite distance, which is unwanted for local reflections. By specifying a bounding box however it is possible to offset the reflection vector so that reflections look correct for local areas [6].

There are three dynamic reflection probes in the scene, each with a cubemap resolution of 256. Due to performance reasons we update only one probe per frame. We don't use the full deferred pipeline for this update, but a simple forward renderer with a very basic illumination model.

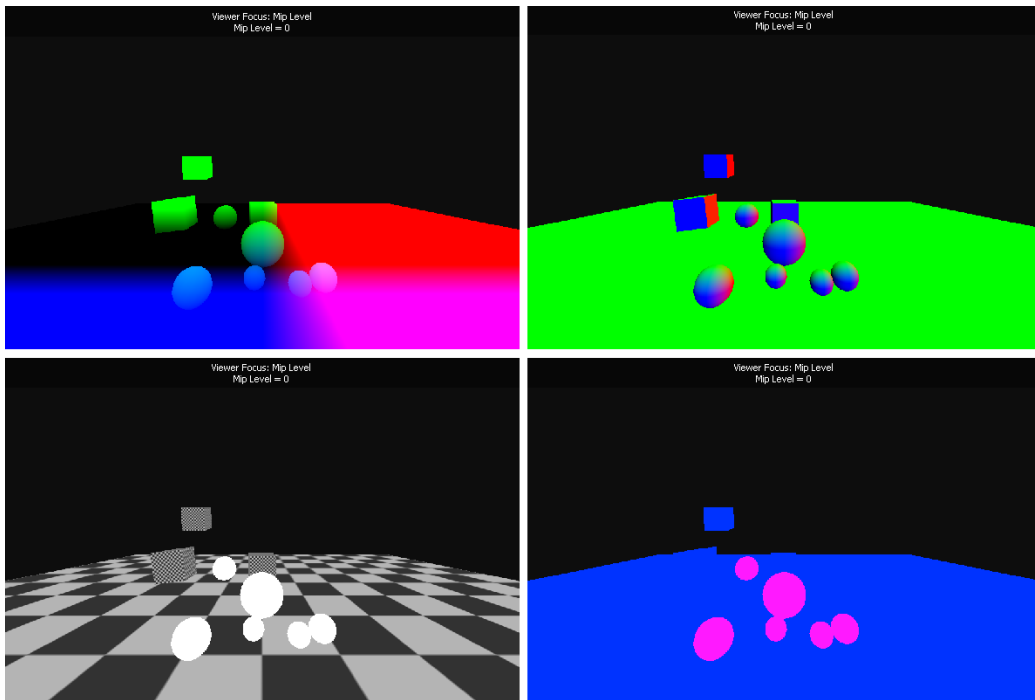


Figure 1: G-Buffer

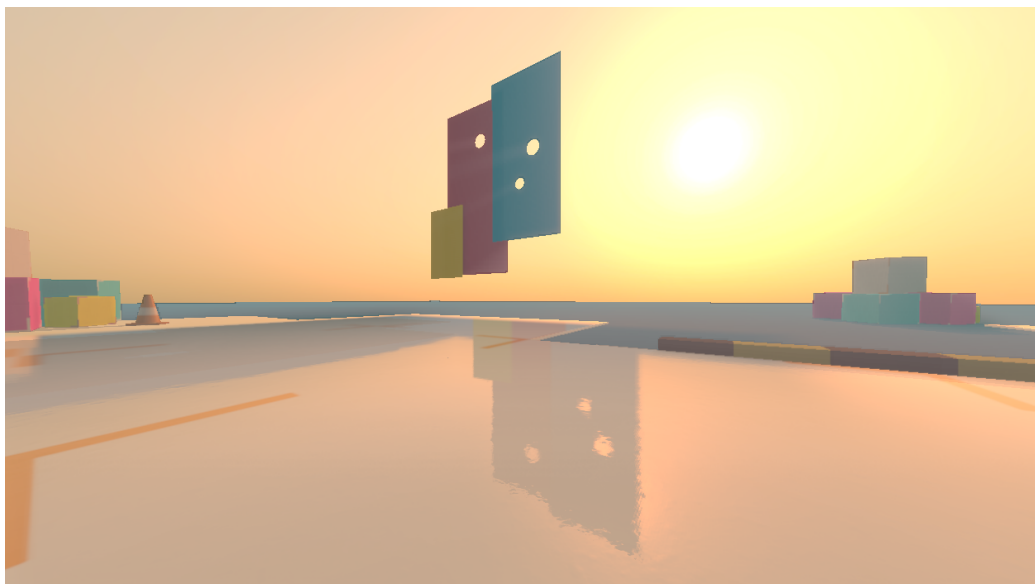


Figure 2: Dynamic reflections

## Image based lighting

Skybox and reflection probes serve as the source for environmental lighting.

Normally you would filter the cubemaps with a BRDF to generate irradiance and radiance maps [5] [7]. We instead rely on a simple hack [9]: we use the highest mipmap level as an irradiance map, and for specular reflections select the mipmap level based on roughness.

## SSDO

For the ambient occlusion effect in our demo we implemented the technique described by Ritschel et al.[12], which is designed to work with image based lighting models like the one our demo uses. We use 64 pairs of sample directions and sample lengths with a maximum radius of 0.2. To reduce the banding artifacts produced by using the same samples for each pixel we use a rotated sample kernel and blur the result of our SSDO pass as described in the LearnOpenGL SSAO tutorial[2] and by Ritschel et al.[12]. By this we get  $4 * 64$  sample sets. We use only 4 different sets so that the blurring step that is used to get rid of the noise pattern produced by reusing the same rotation pattern in every  $4x4$  pixel block is not too strongly visible. Our SSDO step replaces our diffuse lighting contribution in the previous lighting pass and is best visible in the corners of the walls surrounding the bunny.

Ritschel et al.[12] also describe how the technique could be used in a second rendering pass to implement an indirect bounce of light coming from a blocking fragment that is visible in the shadows produced in the first pass. We also implemented this second step but took it out in the end, because it was barely visible.

## Light shafts

Initially we wanted to implement light shafts using the screen space technique described by Mitchell[8], but finally went with the ray marching based technique described in the revision course[3]. We use a shadow map produced by a single directional light that is positioned approximately at the sun's position in the skybox. As described in Realtime Rendering[1], we use three color channels for our medium as well as for the light's color to have more control over the visible outcome of the effect. Our rays are configured to use a step size of 0.1. As described in the revision course[3] and this blog post by Angelo Pesce[10] we downsample our gBuffer to have the resolution used in the other steps using a checkerboard pattern. When upsampling the result back to full resolution we always pick the sample where the total in

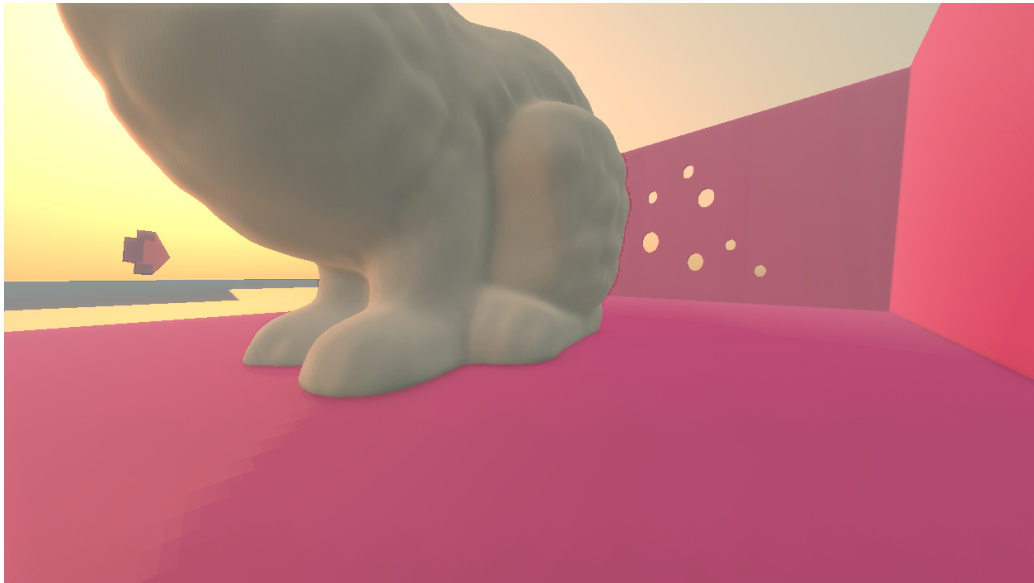


Figure 3: SSDO

depth to the center of the full resolution texture is lowest. Additionally we use a uniform value to control the strength of the volumetric light in the final image.

## Bloom

The basic algorithm is described in [4]. Pixels above a certain threshold get rendered to a texture, this texture gets blurred in two passes (horizontally and vertically). A final pass then blends this texture onto the final scene.

We had problems with single bright pixels, causing the bloom to flicker heavily. We eliminate those single pixels by applying a median filter.

## HDR and tonemapping

The skybox uses HDR textures and therefore drives all the HDR lighting in the scene.

The tonemapping algorithm is equation 4 from Reinhard’s paper [11] and happens in two passes: the luminance for each pixel is rendered into a texture, this texture then gets mipmapped and we can sample the average luminance of the scene from the lowest mip-level.

We do all our lighting calculations in linear space and convert back to gamma as a last step. Textures are converted from sRGB to linear on load.

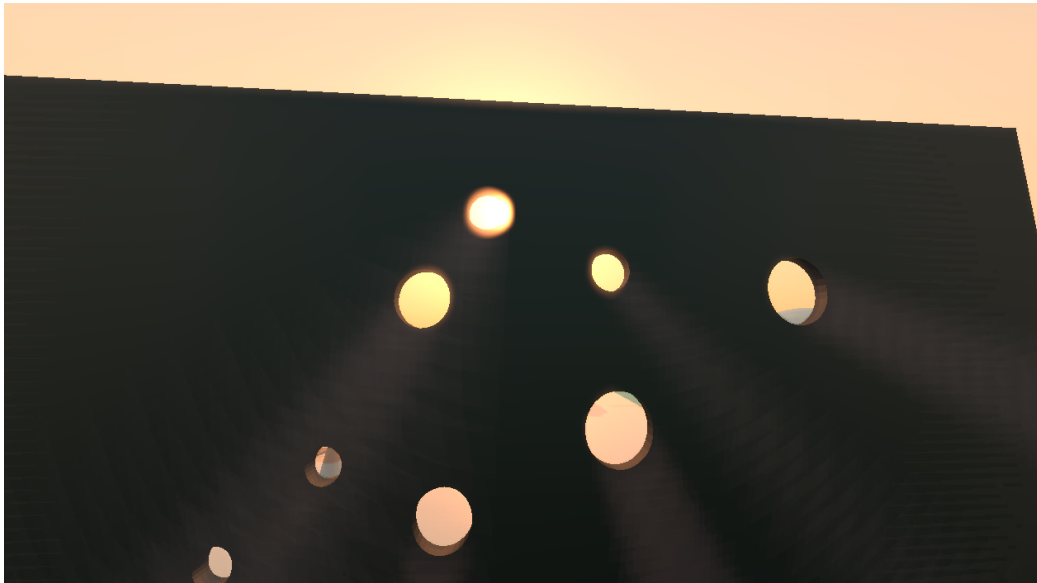


Figure 4: Bloom and light shafts

## Misc information

The demo was tested on NVIDIA graphics card.

The whole scene was created in Blender, including camera animation (based on splines, baked before export). Additional source for models and textures was the Unity Asset Store.

## Used libraries

- Assimp  
FBX loading, <http://www.assimp.org/>
- FMOD  
Audio, <https://www.fmod.com/>
- FreeImage  
Loading images, <http://freeimage.sourceforge.net/>
- PhysX  
All physics related stuff, <https://developer.nvidia.com/physx-sdk>

## References

- [1] Tomas Akenine-Moller, Tomas Moller, and Eric Haines. *Real-Time Rendering*. A. K. Peters, Ltd., Natick, MA, USA, 2nd edition, 2002.
- [2] Joey de Vries. Ssao. <https://learnopengl.com/Advanced-Lighting/SSAO>. Accessed: 2019-01-21.
- [3] Robert Fischer. Volumetric light. <https://tuwel.tuwien.ac.at/course/view.php?id=15757>, 2018. Accessed: 2019-01-21.
- [4] John O'Rourke Greg James. Real-time glow. In Randima Fernando, editor, *GPU Gems*. Addison-Wesley, 2004.
- [5] Gary King. Real-time computation of dynamic irradiance environment maps. In Matt Pharr, editor, *GPU Gems 2*, pages 167–176. Addison-Wesley, 2005.
- [6] Sébastien Lagarde. Image-based lighting approaches and parallax-corrected cubemap. <https://seblagarde.wordpress.com/2012/09/29/image-based-lighting-approaches-and-parallax-corrected-cubemap/>. Accessed: 2019-01-21.
- [7] Josiah Manson and Peter-Pike Sloan. Fast Filtering of Reflection Probes. *Computer Graphics Forum*, 2016.
- [8] Kenny Mitchell. Volumetric light scattering as a post-process. In Hubert Nguyen, editor, *Gpu Gems 3*. Addison-Wesley Professional, first edition, 2007.
- [9] James Wilcox Sam Donow Michael Mara Morgan McGuire, Daniel Evangelakos. Plausible blinn-phong reflection of standard cube mip-maps, 2013.
- [10] Angelo Pesce. Low-resolution effects with depth-aware upsampling. <http://c0de517e.blogspot.com/2016/02/downsampled-effects-with-depth-aware.html>, 2016. Accessed: 2019-01-21.
- [11] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *ACM Trans. Graph.*, 21(3):267–276, July 2002.

- [12] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, pages 75–82, New York, NY, USA, 2009. ACM.