ENSIMAG

# 3D Graphics Projet

---

---

**Achieved by :**

Loginova Aleksandra
Er-rabhi Imane
Granier Bérenger

Academic year : 2024/2025

# Contents

# 1 Introduction

In this project we implemented environment mapping to improve the rendering pipeline seen in the practical sessions of the 3D graphics course. Environment mapping allows more realistic illumination of specular surfaces as in metallic objects, for example. It is the most efficient way to simulate the light that arrives from all directions(which is the case for real world's scenes) rather than from one specific source. Simply put, environment mapping simulates an object reflecting its surroundings.

Figure 1: Chicago "bean" reflecting the environment

# 2 State of the art

Environment mapping is an efficient image-based lighting technique for approximating the appearance of a reflective surface by means of a precomputed texture.
Traditionally, ray tracing has been used for simulating accurate reflections. The main idea behind it is to trace the path of individual light rays as they interact with surfaces in a scene. This approach gave high in fidelity but generally slow results. Therefore, it was mainly deployed in applications where relatively long rendering could be tolerated, such as film visual effects. However, it would not be suitable for real-time applications, such as video games or interactive simulations.
To address these limitations, various approximate methods have been developed. One of the most widely used in interactive computer-graphic rendering is **environment mapping**. Environment mapping is, essentially, a texturing technique. It changes an object's appearance by applying a texture map to its surface. Depending on the object's specularity, the texture would show a higher or lower degree of reflection of the surrounding environment. For instance, objects with low specularity can be textured with an image that approximates the radiance coming from the surrounding environment, instead of mapping it directly as we would do for objects with high specularity.

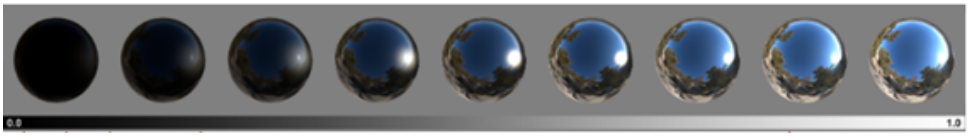


Figure 2: Different levels of specularity for a sphere

The most common type of an environment map is **cube map**. Although, other forms

exist, for example, sphere map. The key assumption we make in environment mapping is that the light captured by an environment map is emitted from infinitely far away. So an environment map consists of directional light sources. There are certain advantages as to using a cube map: it allows for lower resolution maps, due to less distortion as there is more even texel sample density in a cube map. It works better for real-time simulations. Modern real-time engines, including Unreal Engine and Unity, extensively use environment mapping to simulate reflections on materials such as metal, glass, or polished surfaces. These reflections, while not physically accurate in all cases (e.g., they do not reflect nearby objects or self-reflections), provide a visually convincing approximation that performs well in dynamic scenes. Environment mapping is often used in conjunction with Physically Based Rendering (PBR) workflows, where materials are defined by parameters like roughness and metalness, and reflections are blended with lighting models for added realism.

More recently, hybrid approaches have emerged that combine screen-space reflections (SSR), precomputed reflection probes, and ray tracing (when hardware permits) to bridge the gap between performance and realism. However, environment mapping remains a key component of the real-time rendering pipeline due to its simplicity, efficiency, and visual impact.

# 3 Methodology

## 3.1 Cube map

In our implementation, the technique of environment mapping relies on projecting the environment on a cube first, before looping on the polygons of the reflective shape, and for each polygon sampling the part of the cube it's facing. Each side of the scene is projected on a cube face accordingly.

Figure 3: Cube map of the environment used in the project

## 3.2 Skybox rendering

Before rendering geometry, a skybox is rendered using the renderSkybox() method. This is done by iterating over all pixels in the image and shooting a ray from the camera origin

through the far plane, using inverse projection and view matrices. This step allows for a realisting view of the environment from where the camera is located.

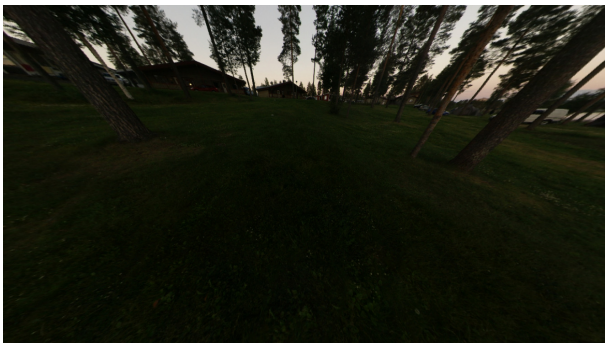


Figure 4: Skybox for the given cube map and camera position

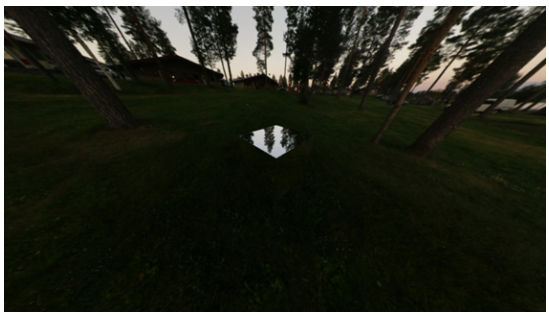
## 3.3 Convertion of coordinates

To calculate which part of the environment will be reflected on which part of the object we calculate the **reflection vector** using the following formula:

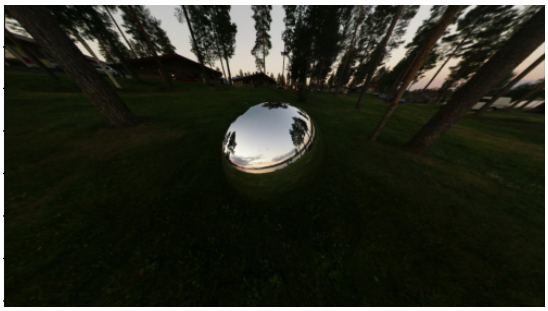
$$\vec{R} = 2(\vec{N} \cdot \vec{V})\vec{N} - \vec{V}$$

Where $\vec{N}$ is the surface normal and $\vec{V}$ is the view direction.

Once the reflection direction $\vec{R}_e$ is computed, we convert it to appropriate cubemap coordinates using the `convert_to_cubemap_coords()` function. This function selects the dominant axis of the direction vector to determine which cubemap face to sample from (posx, negx, posy, etc.), and calculates the corresponding 2D texture coordinates $(u, v)$ within that face. This environment mapping logic is embedded directly into the fragmentShader, replacing the traditional surface shading output.



(a) A cube with environment mapping



(b) A sphere with environment mapping

Figure 5: Environment mapping on specular objects

## 3.4 Decomposition

In order to simulate realistic lighting and reflections in real-time rendering, we must often break down complex reflection models into simpler, more manageable components. This process is known as decomposition.

The most common form of decomposition is the separation of the reflection model into two distinct terms: a diffuse component and a specular component. Each term can

then be treated individually, using appropriate techniques such as hardware lighting or precomputed environment maps.

To go further, especially when dealing with non-metallic surfaces, the specular term can be factored into two parts:

- An environment map (which encodes the incoming light),
- And an angular-dependent weighting factor, usually referred to as the Fresnel term.

The Fresnel term models how reflectivity changes depending on the angle of incidence and the optical density (index of refraction) of the material. For metallic surfaces, this term is close to 1 and mostly constant. However, for non-metallic materials, the Fresnel term varies significantly with angle and cannot be neglected.

To account for angle-dependent reflectivity, we use a simplified model of the Fresnel effect. Instead of evaluating the full Fresnel equations—which require complex calculations involving the index of refraction—we adopt Schlick's approximation:

$$F(\theta) = F_0 + (1 - F_0)(1 - \cos\theta)^5$$

where $\theta$ is the angle between the view vector and the surface normal, and $F_0$ is the base reflectivity at normal incidence. This expression captures the main visual behavior of the Fresnel effect with much lower computational cost.

In our implementation, we directly set $F_0$ to fixed values depending on the material type:

- $F_0 = 0.04$ for dielectric materials (e.g., plastics),
- $F_0 = 0.1$ for moderately reflective surfaces,
- $F_0 = 1.0$ for metallic surfaces.

This allows us to modulate the contribution of the environment map (which encodes specular reflections) depending on the view direction. We blend the reflected color and the local shading using Schlick's approximation:

$$L_o = F \cdot L_m + (1 - F) \cdot L_d$$

Where:

- $L_o$ is the outgoing radiance,
- $L_m$ is the mirror reflection sampled from the environment map,
- $L_d$ is the local diffuse and specular shading from the Phong model,
- $F$ is the Fresnel term computed via Schlick's approximation.

The following figures illustrate the impact of different values of $F_0$ on the appearance of the specular reflection:

(b) Environment mapping with $F_0 = 0.04$ (dielectric material)

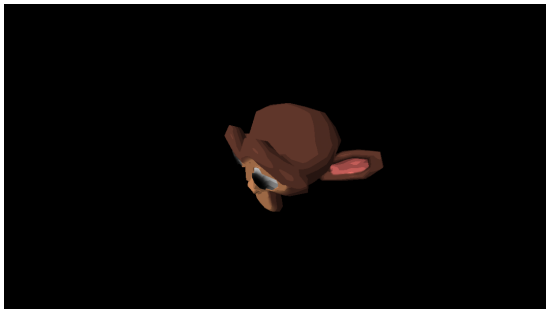
(c) Environment mapping with $F_0 = 0.1$ (more reflective surface)

(d) Environment mapping with $F_0 = 1.0$ (metallic surface)

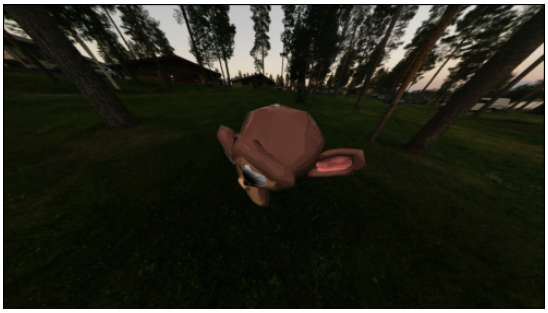
Figure 6: Effect of the Fresnel term $F_0$ on the specular component in real-time rendering.

# 4 Evaluation of the method

The method successfully produces 3D images from scratch using a software implementation. Despite the absence of hardware acceleration or external libraries, the renderer achieves visually coherent results with smooth shading and realistic lighting. The average rendering time per frame is approximately 20 seconds, which we consider to be a low computational cost given the context of a CPU-based educational implementation. This demonstrates a good trade-off between visual quality and performance for the scope of the project.



(b) No environment Mapping



(c) With Environment mapping

# 5 Limitations

One limitation of the current environment mapping implementation lies in its reliance on static cubemap sampling based solely on the reflection vector. While this approach effectively simulates reflections from a distant environment, it assumes the scene is surrounded by an infinitely far environment, which neglects local geometry and occlusion effects. This can result in unrealistic reflections, especially for objects close to other geometry, since they do not block or alter the reflected view.

Moreover, this method does not support dynamic updates to the environment (e.g. moving lights or objects), leading to static and sometimes inconsistent visuals. In our implementation, the execution time is approximately 13 seconds and might struggle with real-time applications. To address these issues, more advanced techniques such as screen space reflections (SSR) or ray-traced reflections can be used. SSR leverages the rendered

image to compute approximate reflections of nearby geometry in real time, providing more accurate and dynamic reflections with low overhead, though it suffers from visibility limitations outside the screen. Ray tracing, on the other hand, can simulate physically correct reflections and occlusions at the cost of significantly higher computational complexity, but modern GPUs and APIs (e.g. DirectX Raytracing or Vulkan RT) are increasingly capable of supporting it in real-time rendering pipelines. Another limitations is that, for non-convex shapes, in real life some parts of the environment appear on several parts of the shape, which is not possible to do with environment mapping.

# References


[1] Wikipedia contributors. *Reflection mapping*. Available at: `https://en.wikipedia.org/wiki/Reflection_mapping`,

[2] Simone Kriglstein, Günter Wallner. *E*nvironment mapping. Available at: `https://www.cg.tuwien.ac.at/courses/Seminar/SS2001/envmap/environment_mapping.pdf`,

[3] Various sources. *Environment mapping*. Available at: `https://www.sciencedirect.com/topics/computer-science/environment-mapping`,

[4] Tom Thorne. *Computer Graphics 9 - Environment mapping and mirroring*. Available at: `https://www.inf.ed.ac.uk/teaching/courses/cg/lectures/slides9.pdf`,

[5] University of California San Diego. *Environment mapping*. Available at: `https://cseweb.ucsd.edu/classes/wi18/cse167-a/lec13.pdf`,

[6] University Lyon 1. *Affichage d'un environnement*. Available at: `https://perso.univ-lyon1.fr/jean-claude.iehl/Public/educ/M1IMAGE/html/group__cubemap.html`