

**ALMA MATER STUDIORUM · UNIVERSITÀ DI  
BOLOGNA**

---

**Corso di Laurea Magistrale in  
Ingegneria Informatica**

**SHADER: Texture mapping,  
Normal mapping,  
Environmental & Refraction  
LAB 7**

Fondamenti di Computer Graphics M

**Cristian Davide Conte 0001034932**

**Anno Accademico: 2022-2023**

# Contents

<b>1</b>	<b>Camera Motion</b>	<b>2</b>
<b>2</b>	<b>Texture mapping 2D</b>	<b>3</b>
<b>3</b>	<b>Texture mapping 2D + Shading</b>	<b>4</b>
<b>4</b>	<b>Procedural mapping</b>	<b>5</b>
<b>5</b>	<b>Normal Mapping</b>	<b>6</b>
<b>6</b>	<b>Environment cube mapping: skybox</b>	<b>7</b>
<b>7</b>	<b>Environment mapping: object REFLECTION</b>	<b>8</b>
<b>8</b>	<b>Environment mapping: object REFRACTION</b>	<b>9</b>
<b>9</b>	<b>Oggetti semi-trasparenti</b>	<b>10</b>

# **Chapter 1**

## **Camera Motion**

Il movimento della camera lungo un percorso è stato creato riutilizzando le implementazioni delle curve di Bezier dell'esercitazione 1. In particolare, alla pressione del tasto "k", l'applicazione salverà la posizione attuale della camera e la utilizzerà successivamente come control point (qui chiamati "keyframe") di una curva di Bezier (approssimante via algoritmo di deCasteljau) lungo la quale la camera si muoverà autonomamente: il control point finale viene forzato ad essere coincidente con quello iniziale per avere una curva chiusa. Per rimuovere l'ultimo keyframe basterà premere "l", mentre per far partire/interrompere l'animazione della camera è necessario premere la barra spaziatrice. Esattamente come nelle curve di Bezier, è necessario che almeno due control point siano presenti al fine di creare una curva.

A questo link è presente una breve demo del movimento della camera lungo una curva di Bezier:  
[https://github.com/CristianDavideConte/Fondamenti-di-Computer-Graphics/blob/main/LAB\\_07/Camera%20path%20demo.mp4](https://github.com/CristianDavideConte/Fondamenti-di-Computer-Graphics/blob/main/LAB_07/Camera%20path%20demo.mp4)

# Chapter 2

## Texture mapping 2D

Gli shader "v\_texture.gsl" e "f\_texture.gsl" mappano le coordinate della texture presente nel file "WoodGrain.bmp" ad ogni vertice dell'oggetto "torus" tramite la funzione "loadTexture". Lo shading del toro è stato cambiato in "ShadingType::TEXTURE\_ONLY".

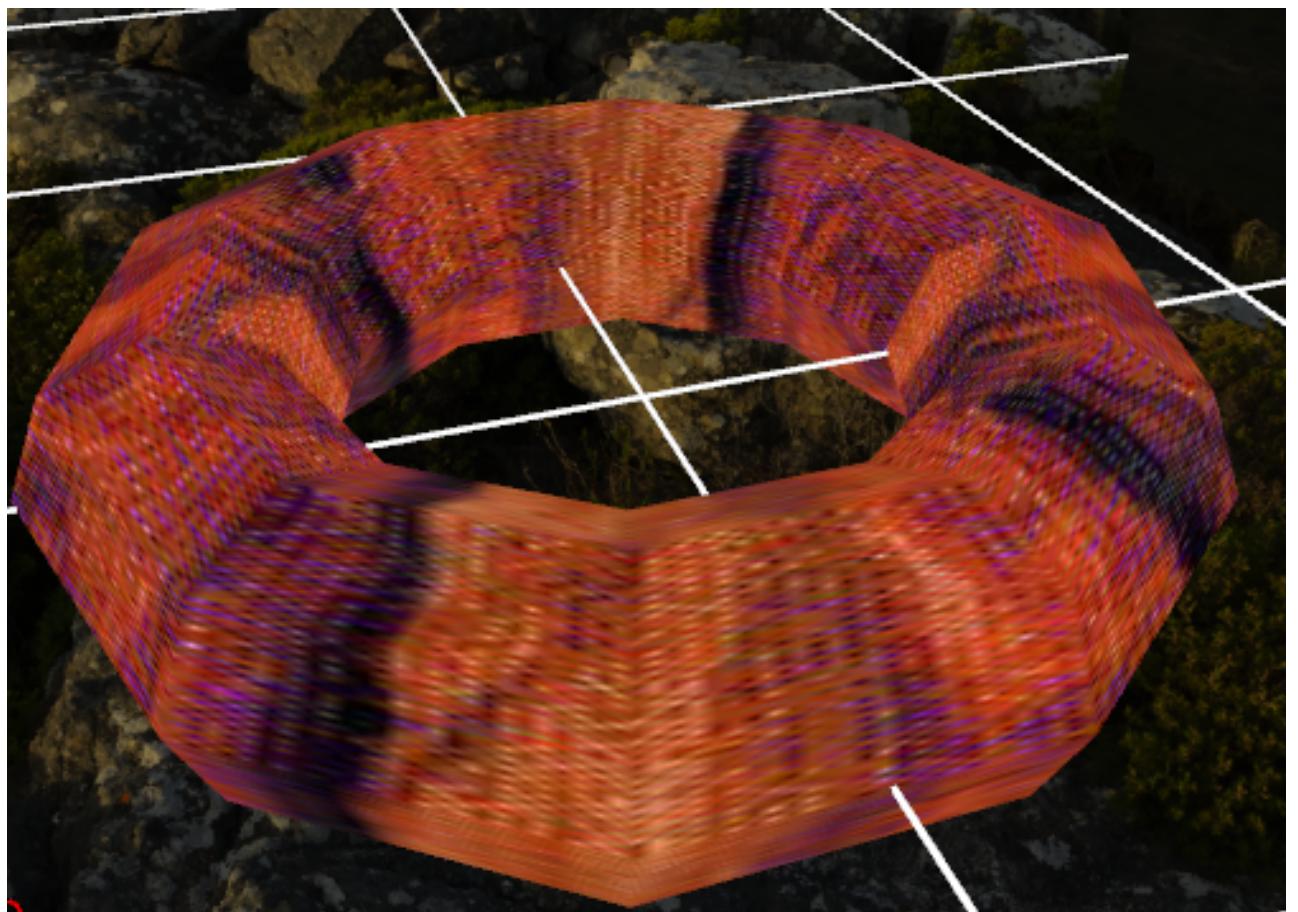


Figure 2.1: Torus - Texture 2D

# Chapter 3

## Texture mapping 2D + Shading

Per includere il modello di illuminazione di Phong all'interno degli shader precedenti sono state create due varianti chiamate "v\_texture\_phong.glsl" e "v\_texture\_phong.glsl" che seguono quanto fatto nell'esercitazione 3.

Lo shading del toro è stato cambiato in "ShadingType::TEXTURE\_PHONG".



Figure 3.1: Torus - Texture 2D + Shading

# Chapter 4

## Procedural mapping

Al fine di generare una texture procedurale, si è prima cercato di riutilizzare lo stesso algoritmo di generazione del pattern checkerboard della scacchiera presente in scena e successivamente si è confrontato il risultato con una noise texture generata tramite algoritmo di Perlin.

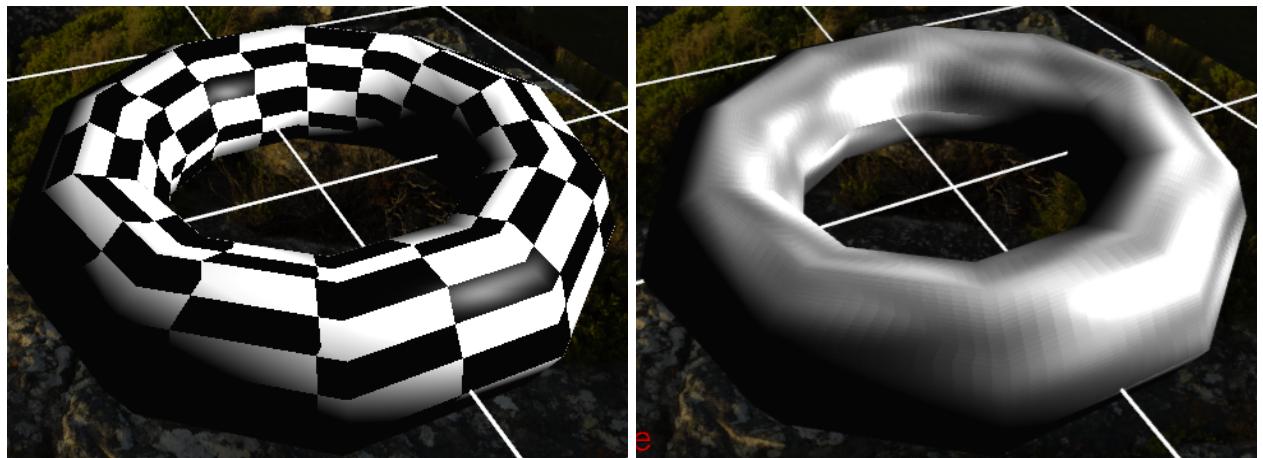


Figure 4.1: Torus - Checkerboard texture (sx) vs Perlin texture (dx)

# Chapter 5

## Normal Mapping

All'interno degli shader "v\_normal\_map.glsl" e "f\_normal\_map.glsl" vengono combinati sia il texture mapping, sia il modello di illuminazione di Phong che una normal mapping: quest'ultima modifica il modo in cui la luce interagisce con l'oggetto (modifica i vettori normali alla superficie) simulando la profondità.

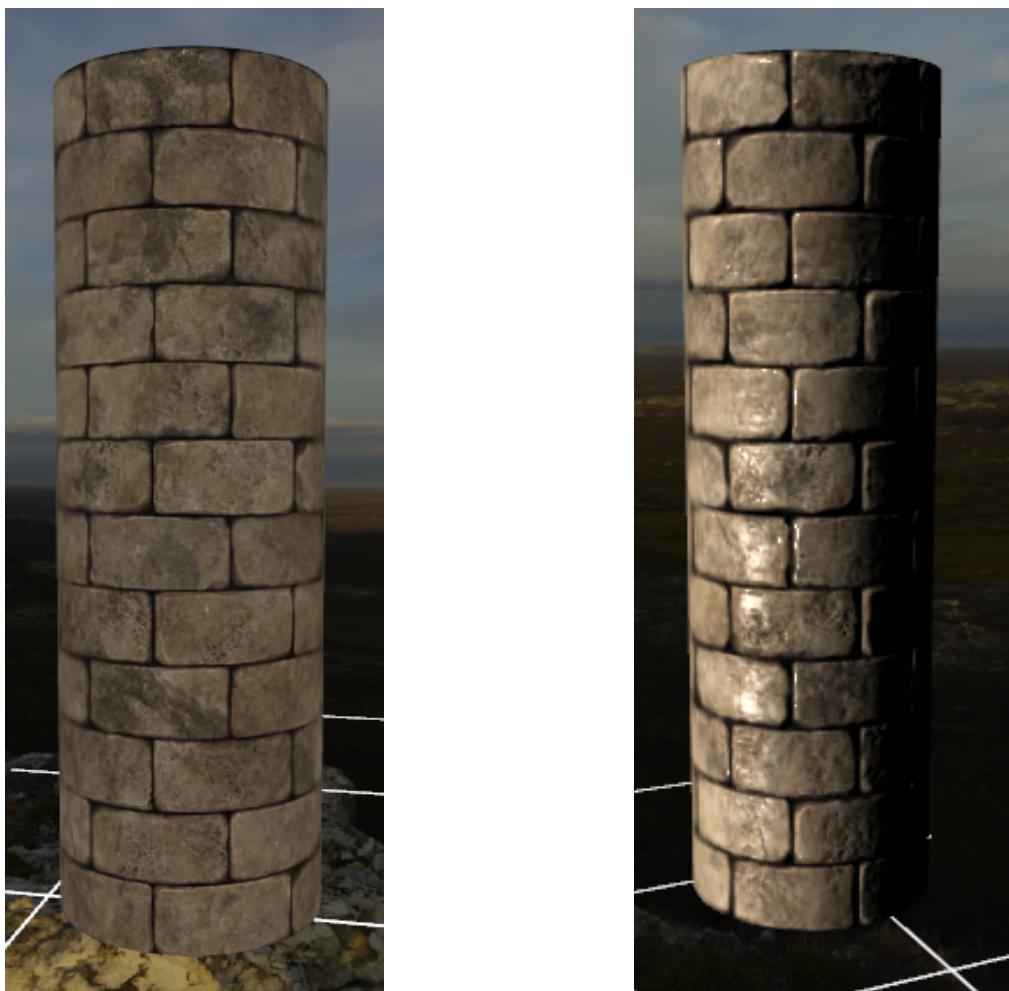


Figure 5.1: Colonna - Texture Mapping (sx) vs Normal Mapping (dx)

# Chapter 6

## Environment cube mapping: skybox

Il cube mapping è realizzato tramite gli shader "v\_skybox\_advanced.glsl" e "f\_skybox.glsl" che permettono di mappare le texture dello skybox su un cubo. È inoltre stata modificata la posizione dello skybox in modo che contenga la scena (traslazione della matrice M dell'oggetto).

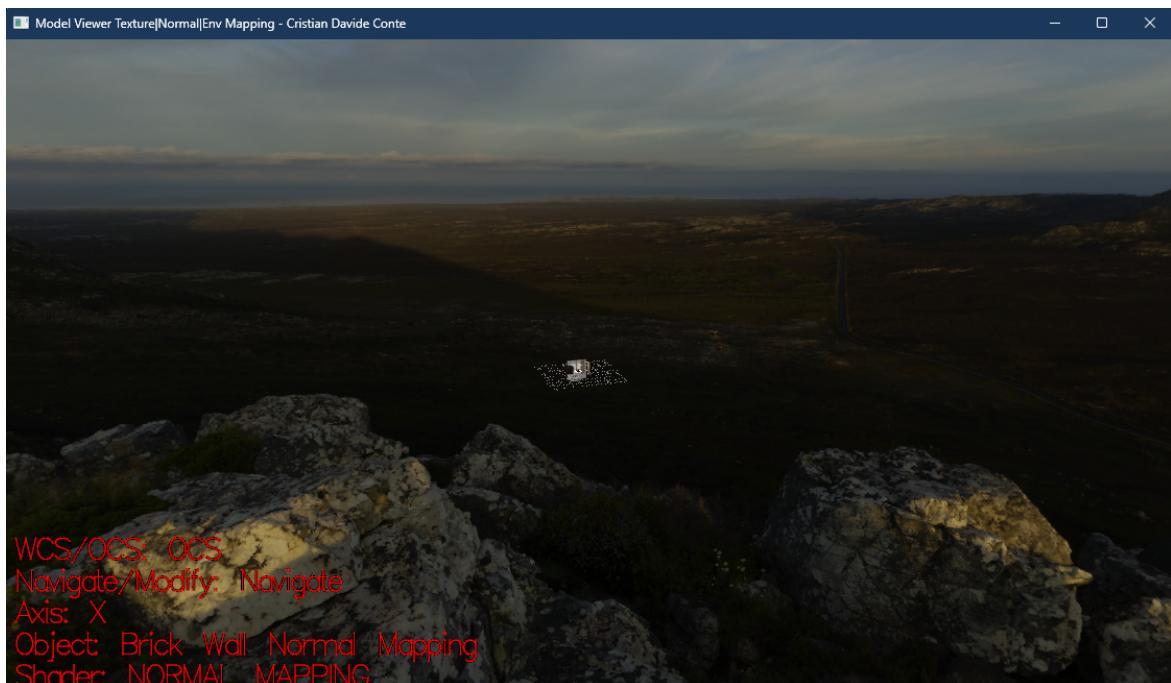


Figure 6.1: Environment mapping - Skybox

# Chapter 7

## Environment mapping: object REFLECTION

Come già fatto per lo skybox, anche per la sfera presente in scena è stato applicato l'environment mapping. Lo shader di quest'ultima è stato impostato su "ShadingType::REFLECTION" in modo che possa riflettere la luce della scena circostante.

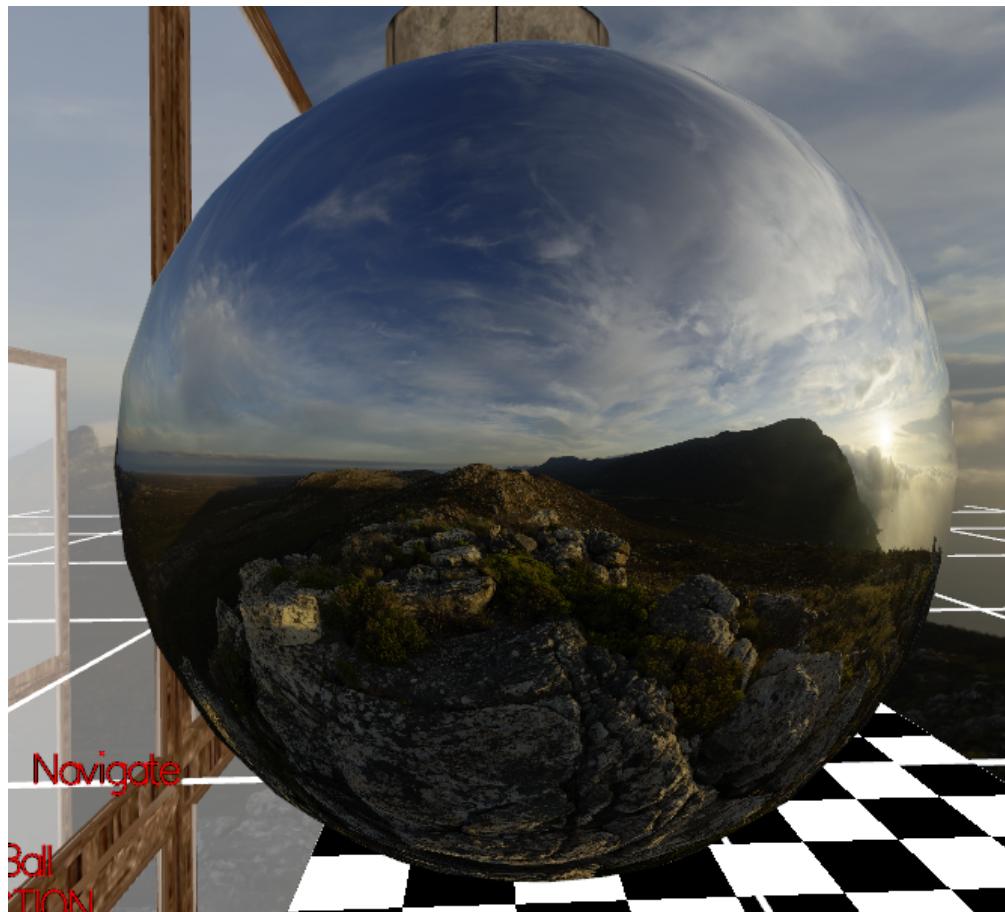


Figure 7.1: Environment mapping - Sfera + Reflection

# Chapter 8

## Environment mapping: object REFRACTION

Come già fatto per lo skybox e la sfera, anche per il cubo presente in scena è stato applicato l'environment mapping. Lo shader di quest'ultima è stato impostato su "ShadingType::REFRACTION" in modo che possa rifrangere la luce della scena circostante. A differenza della riflessione, in questo caso il cubo non specchia lo skybox ma fornisce una visione "distorta" di quello che c'è oltre il cubo.

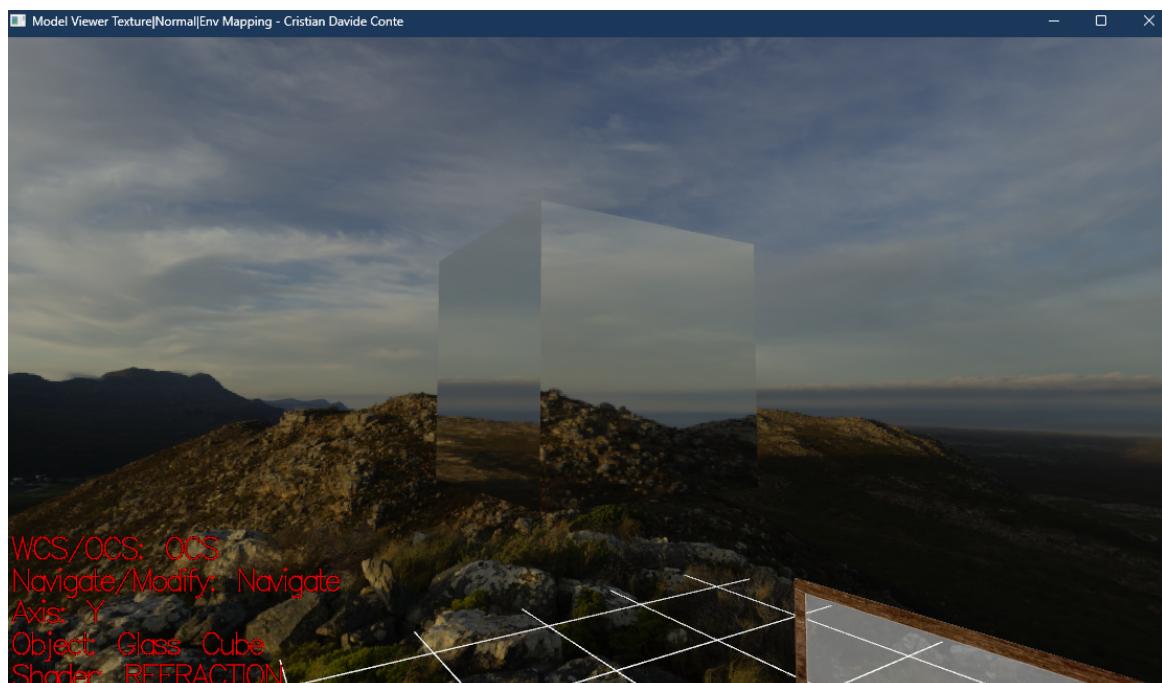


Figure 8.1: Environment mapping - Cubo + Refraction

# Chapter 9

## Oggetti semi-trasparenti

L'algoritmo per la gestione di oggetti trasparenti utilizzato funziona in questo modo:

- Gli oggetti in scena sono stati divisi in opachi e semi-trasparenti
- Sono stati disegnati gli oggetti opachi
- È stato disabilitato lo z-buffer
- Gli oggetti semi-trasparenti sono stati ordinati per distanza dalla camera (dal più lontano al più vicino)
- Sono stati disegnati gli oggetti con trasparenze
- È stato ri-abilitato lo z-buffer

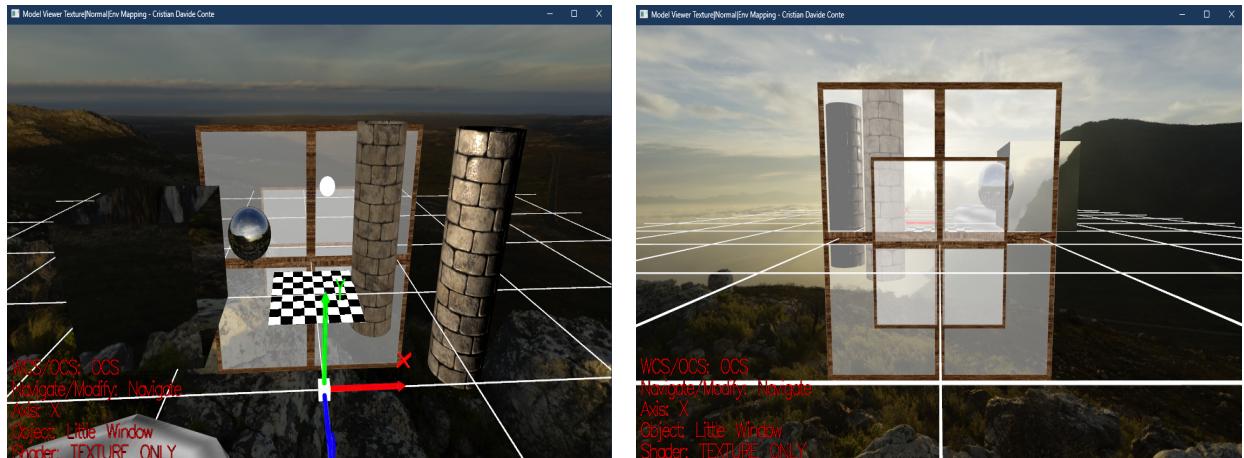


Figure 9.1: Resa di oggetti semi-trasparenti