Report di Computer Graphics

Andrea Schinoppi - matricola 0001097628

A.A. 2022/2023

Indice

1 Laboratorio 1

Scopo di questo laboratorio era modificare il codice fornito per:

1. Disegnare la curva di Bézier a partire dai punti di controllo inseriti, utilizzando l'algoritmo di de Casteljau;
2. Permettere la modifica della posizione dei punti di controllo tramite trascinamento con il mouse;
3. Integrare nel programma il disegno di una curva di Bézier interpolante a tratti (Catmull-Rom Spline).

1.1 Istruzioni

Premendo con il tasto sinistro del mouse sull'area del programma è possibile posizionare a proprio piacimento dei punti di controllo.
Premendo `s` è possibile visualizzare le linee che collegano i punti di controllo.

Una volta inseriti almeno tre punti di controllo sarà possibile visualizzare la relativa curva di Bézier e la Catmull-Rom spline.
Premendo il tasto `v` sarà possibile visualizzare i punti di controllo della spline.

Avvicinandosi ad un punto e premendo il tasto sinistro del mouse invece di inserire un nuovo punto sarà possibile spostare tramite trascinamento il punto scelto.
I tasti `f` ed `l` rimuovono rispettivamente il primo e l'ultimo punto inseriti.

1.2 Screenshot

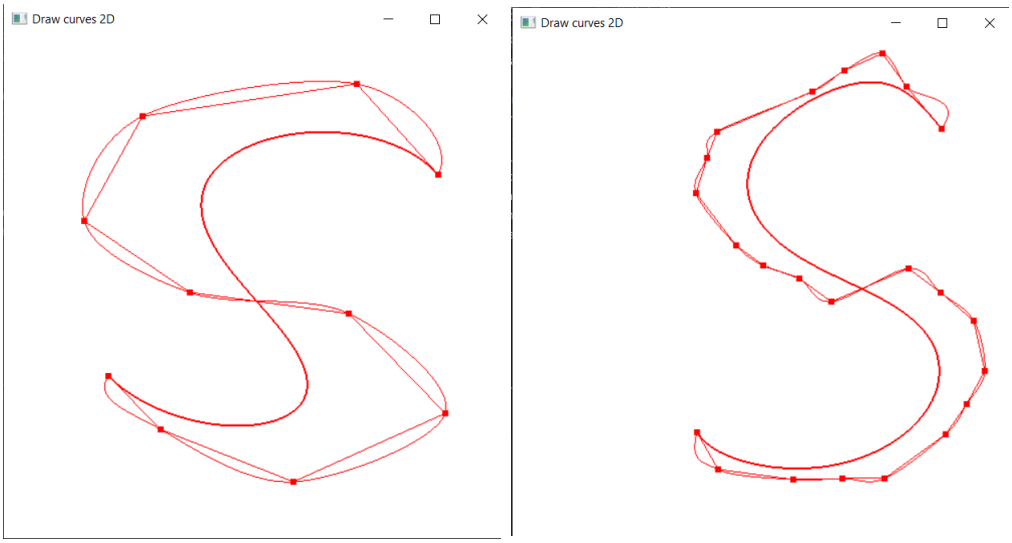

Figura 1.1: Curve di Beziér e Catmull-Rom spline con diversi numeri di punti

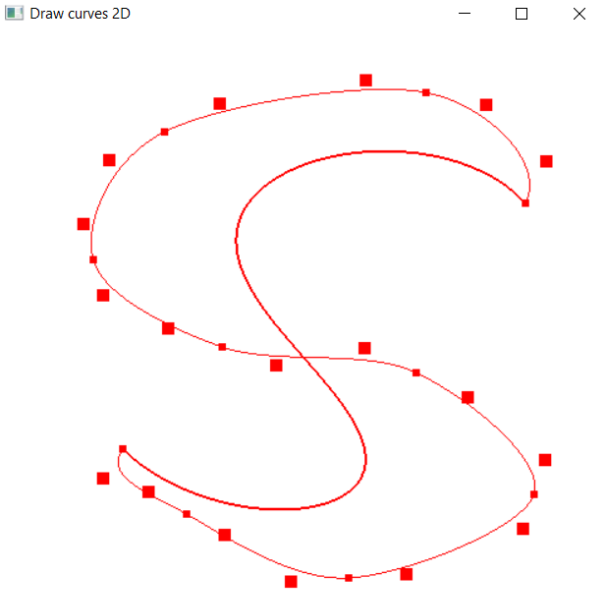

Figura 1.2: Catmull-Rom Spline con punti intermedi

1.3 Curva di Bézier

Il calcolo dei punti della curva di Bézier è affidato all'algoritmo di de Casteljau, implementato nell'omonima funzione `decasteljau`.
Questa funzione viene chiamata con valori di `t` compresi tra 0 e 1 con incrementi di 0.01 ogni volta (100 punti) e applica l'algoritmo sui punti di controllo scelti dall'utente, restituendo le coordinate dei punti della curva di Bézier.

1.4 Modifica della posizione dei punti

Per implementare questa funzionalità sono state sfruttate le callback `glutMotionFunc` e `glutPassiveMotionFunc` che reagiscono al movimento del mouse rispettivamente quando un tasto è premuto e quando nessun tasto è premuto.
Tuttavia la funzione associata alla PassiveMotion verifica che sia premuto il tasto sinistro del mouse per impostare il booleano che indica se si sta trascinando il mouse a true. Questo per non controllare trascinamento e spostamento dei punti con la stessa callback.

Se l'utente sta trascinando il mouse sufficientemente vicino ad un control point, esso sarà agganciato e spostato fintanto che l'utente non lascerà il tasto sinistro del mouse.

1.5 Catmull-Rom spline

Per disegnare la Catmull-Rom Spline, viene applicato l'algoritmo di Catmull-Rom, generando due punti (P^+ e P^-) intermedi per ogni coppia di punti di controllo (P_i e P_{i+1}), stimando la derivata m_i in P_i e calcolando
$P^+ = P_i + \frac{1}{3}m_i$ e $P^- = P_i - \frac{1}{3}m_i$.
A questo punto viene calcolata la curva di Beziér passante per i quattro punti precedentemente descritti.

Ogni curva viene definita da 50 punti in modo che risulti sufficientemente morbida ma che non impieghi troppe risorse per essere calcolata.

2 Laboratorio 2

Scopo di questo laboratorio era creare una demo di animazione digitale 2D interattiva.
È stato realizzato un semplice videogioco chiamato `TriangleRain` in cui il giocatore, rappresentato da un quadrato giallo, dovrà schivare dei triangoli che cadono dall'alto rappresentanti delle stalattiti all'interno di una grotta.

2.1 Istruzioni

Scopo del gioco, come anticipato, è schivare i triangoli che cadono dall'alto fino a raggiungere un punteggio di 1000.

I comandi non sono case-sensitive e sono i seguenti:

1. Tasto 'A': sposta il giocatore verso sinistra;
2. Tasto 'D': sposta il giocatore verso destra;
3. Tasto 'P': mette in pausa il gioco;
4. Tasto 'esc': termina la partita ed esce dal gioco.

Se il giocatore viene colpito da un triangolo la partita termina.

2.2 Screenshot

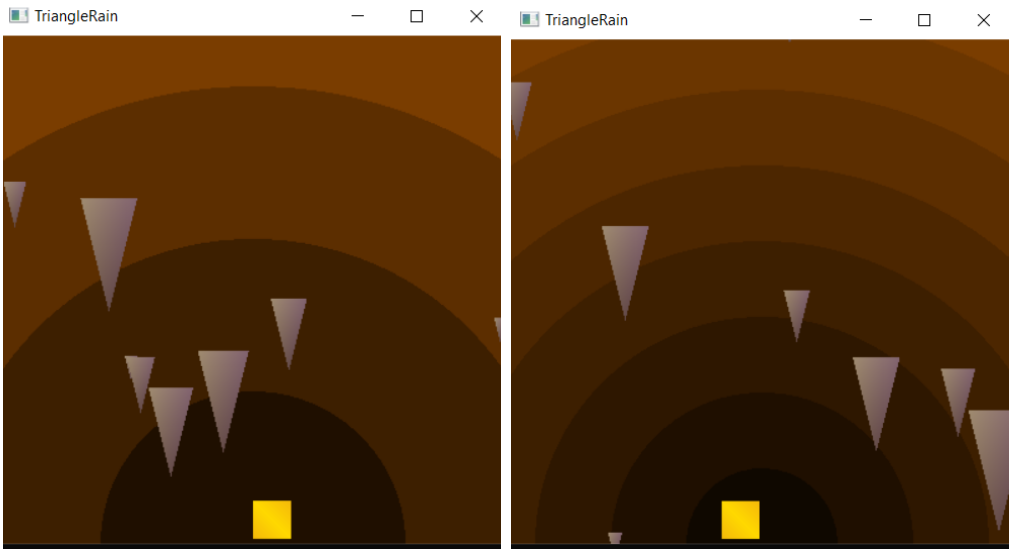

Figura 2.1: Screenshot di gioco a inizio e metà partita

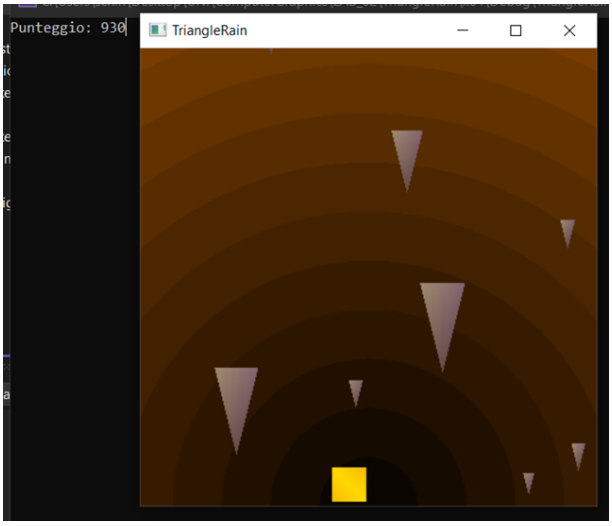

Figura 2.2: Screenshot di gioco nelle ultime fasi della partita

2.3 Implementazione

Prima di giungere alla versione finale, è stato implementato lo stesso gioco con dinamiche simili (presente nella cartella `/OldVersion`) mediante uso di `GlBegin()` e `GlEnd()`.
Tuttavia per mantenere la coerenza con quanto visto a lezione, è stato rifatto da capo utilizzando VAO e VBO.

2.3.1 Grotta

La grotta funge da sfondo al gioco. Essa è implementata creando inizialmente cinque cerchi di tonalità differenti di marrone i cui centri sono leggermente spostati verso il basso.

Proseguendo nel gioco, per dare l'idea che il giocatore si stia avventurando nelle profondità della grotta, il numero di cerchi aumenta (fino ad un massimo di 25), rendendo ogni cerchio meno spesso, ma mantenendo sempre le diverse tonalità di marrone.

2.3.2 Triangoli

I triangoli rappresentano gli ostacoli del gioco: ognuno è descritto all'interno del codice da:

- Le proprie coordinate x e y;
- Una velocità verticale;

- Una velocità orizzontale;
- Larghezza e altezza.

Tutti questi parametri hanno upper-bound definiti da costanti in modo da non poter avere triangoli di dimensioni troppo grandi o piccole o con velocità troppo elevate.
Se durante la caduta un triangolo urta uno dei bordi laterali, la sua velocità orizzontale viene invertita, creando un effetto rimbalzo.

Quando un triangolo supera il bordo inferiore dell'area di gioco significa che è stato schivato e permette al giocatore di ottenere 10 punti.
A seguito dell'uscita dall'area di gioco, il triangolo viene rimosso e ne viene generato uno nuovo completamente casuale al di sopra dell'area.

Infine, i triangoli sono colorati anch'essi di marrone: sono state impiegate tre tonalità differenti di marrone (una per vertice) per creare un effetto sfumatura, prestando attenzione a non creare confusione con il marrone dello sfondo.

2.3.3 Giocatore

Il giocatore è rappresentato da un quadrato giallo in grado di muoversi a destra e sinistra nel campo di gioco.

Sono state utilizzate due tonalità di giallo per rendere il giocatore esteticamente più appagante.

3 Laboratorio 3

Scopo del laboratorio era estendere il codice fornito per implementare:

1. Calcolo e memorizzazione delle normali ai vertici per i modelli mesh poligonali;
2. Un materiale diverso da quelli forniti;
3. L'animazione di un height field mesh modificando la posizione dei vertici in un vertex shader per ottenere l'effetto di un moto ondoso;
4. Il toon shading;
5. La navigazione interattiva in scena;
6. La trasformazione degli oggetti in scena.

3.1 Normali ai vertici

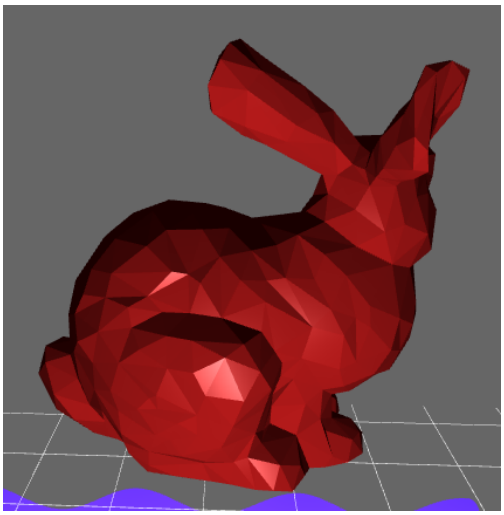

Figura 3.1: Normali alle facce

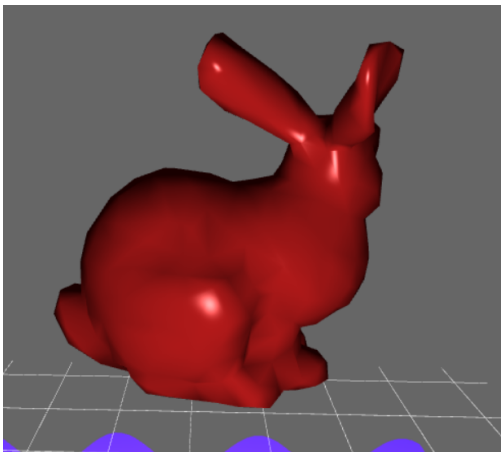

Figura 3.2: Normali ai vertici

3.2 Materiale personalizzato

Data la presenza di un oggetto a forma di coniglio è stato creato un materiale che ricordasse il cioccolato al latte.

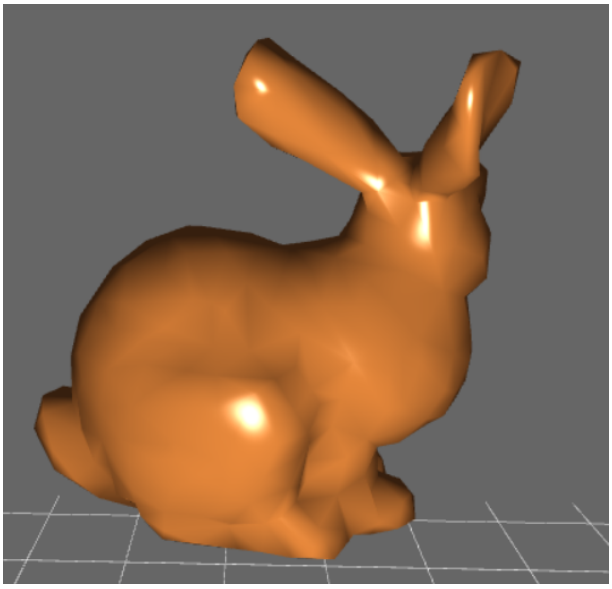

Figura 3.3: Materiale personalizzato

Inoltre, per sperimentare ulteriormente con i materiali, è stata modificata anche l'acqua:

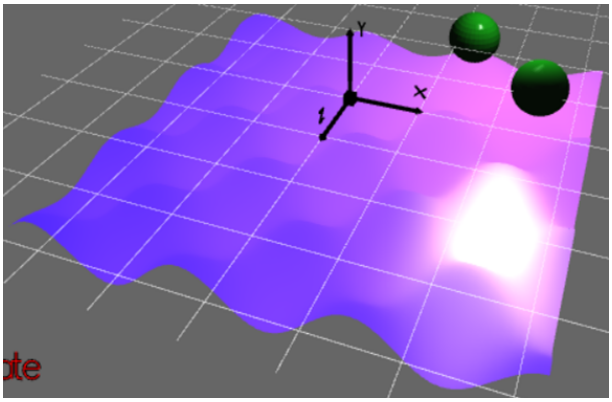

Figura 3.4: Acqua

3.3 Animazione delle onde

Per ottenere l'animazione delle onde sono stati creati vertex e fragment shader dedicati.
Nel vertex shader è stata implementata la formula

$$v_y = \alpha sin(\omega t + 10v_x)sin(\omega t + 10v_z))$$

dove t rappresenta il tempo passato, $\alpha = 0.1$ è l'ampiezza dell'oscillazione e $\omega = 0.001$ la frequenza.

3.4 Toon shading

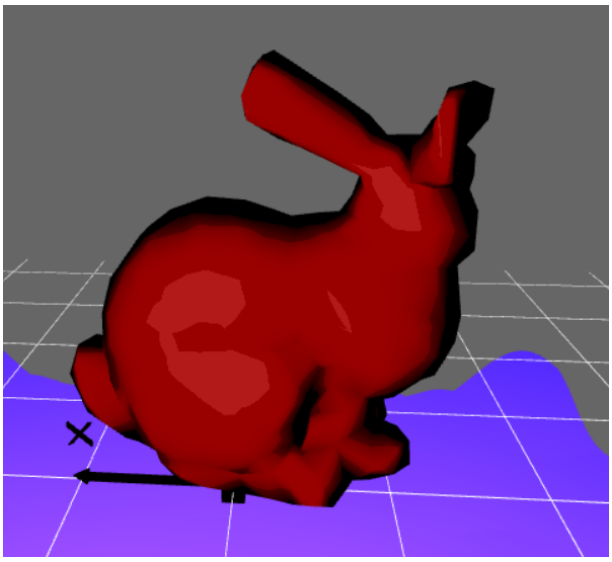

Figura 3.5: Toon shading

Per ottenere il toon shading sono stati creati vertex e fragment shader dedicati.
Nel fragment shader sono stati inseriti cinque livelli di intensità luminosa sulla base dei quali cambia il colore del toon shading.

3.5 Navigazione in scena

Per ottenere il pan orizzontale della telecamera quando l'utente preme control e muove la mouse wheel sono state copiate le funzioni già presenti `moveCameraUp()` e `moveCameraDown()` sostituendo il vettore `upVector` che agisce sulla coordinata y con il vettore `directionVector` che agisce sulla coordinata x in modo che la telecamera trasli sull'asse x, quindi in orizzontale.

3.6 Trasformazione degli oggetti in scena

Per completare quest'ultimo punto è stata implementata la funzione `modifyModelMatrix`.
Per prima cosa vengono create le matrici di trasformazione per il WCS. In seguito viene controllato se la modalità scelta sia OCS, in tal caso le matrici

di trasformazione precedentemente create vengono adattate a questo sistema di coordinate. Per fare ciò, vengono create due matrici per traslare l'oggetto al centro del sistema di coordinate del mondo e per riportarlo nella sua posizione originale
Successivamente viene computata l'operazione da svolgere, in base alla selezione dell'utente.

Infine, se è la luce ad essere mossa, tutti gli shader vengono ricaricati in modo da ottenere l'effetto dello spostamento della fonte luminosa.

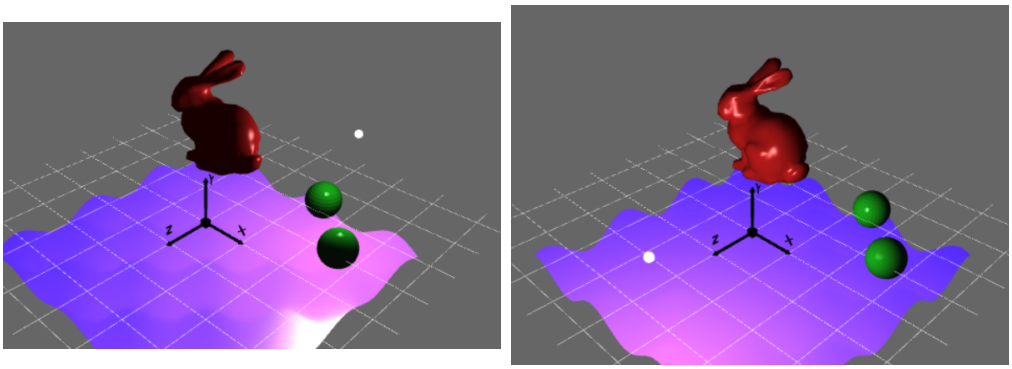

Figura 3.6: Spostamento della luce

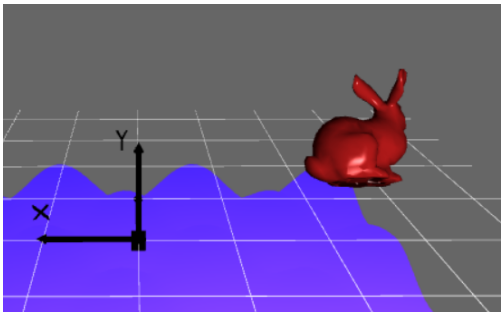

Figura 3.7: Trasformazioni in OCS

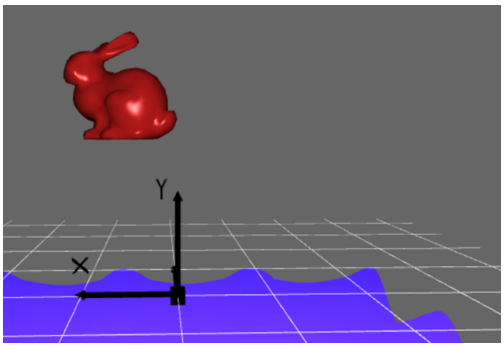

Figura 3.8: Trasformazioni in WCS (Y)

4 Laboratorio 4

Scopo di questo laboratorio è estendere il codice fornito per:

1. Sviluppare la gestione degli shadow rays per generare hard shadows;
2. Sviluppare la gestione ricorsiva dei reflection rays;
3. Implementare una strategia per la gestione di soft shadows mediante risorse luminose ad area anziché puntiformi.

Il programma può essere avviato da linea di comando o tramite lo script `run_LAB_04.bat`.

4.1 Hard shadows

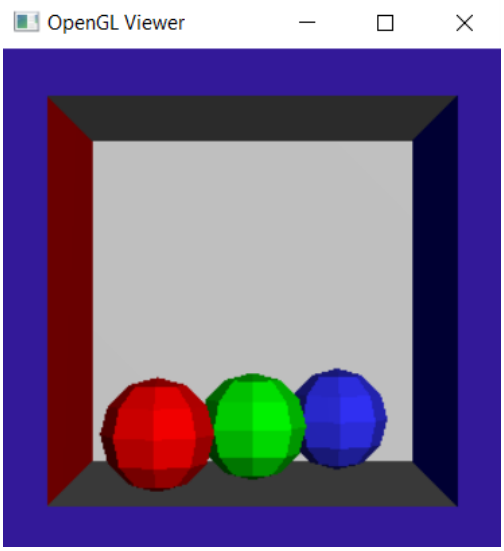

Figura 4.1: Situazione di partenza

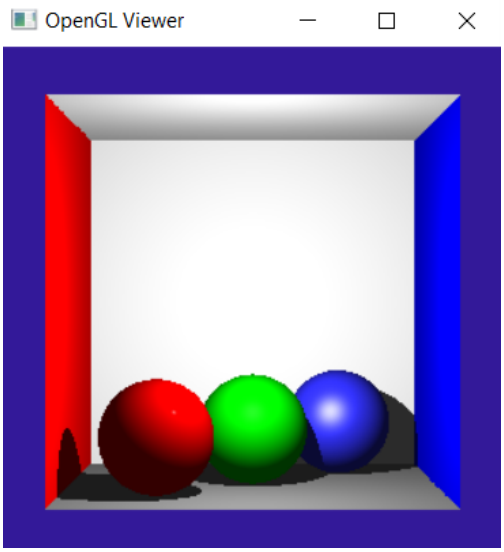

Figura 4.2: Hard shadows

Le hard shadows vengono calcolate lanciando dei raggi mediante la funzione `CastRay` e verificando se ogni raggio interseca un altro oggetto nella scena. Se il raggio d'ombra interseca un altro oggetto, allora la luce non contribuisce alla luminosità del punto sulla superficie.

4.2 Reflection rays

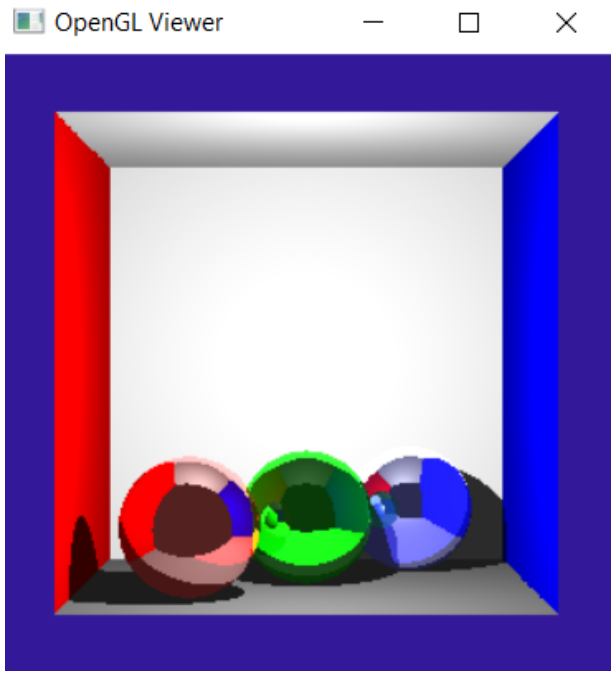

Figura 4.3: Gestione ricorsiva dei reflection rays

Se il numero di bouces è maggiore di 0 e la superficie colpita dal raggio è riflettente, la funzione `TraceRay` calcola il raggio riflesso e si chiama ricorsivamente per tracciarlo.

Il colore restituito dalla chiamata ricorsiva viene moltiplicato per il colore riflettente del materiale e aggiunto al colore dell'oggetto.

4.3 Soft shadows

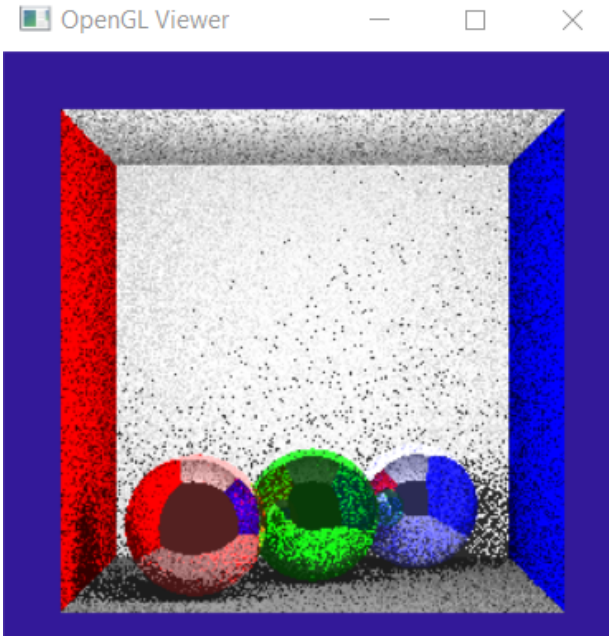

Figura 4.4: Soft shadows

Quando le soft shadows sono attivate, la funzione sceglie casualmente un punto sulla superficie della luce e lancia un raggio verso quel punto.
Questo processo viene ripetuto più volte e il contributo della luce al colore finale viene calcolato come la media dei campioni.

5 Laboratorio 5

Scopo di questo laboratorio era utilizzare il software Blender per modellare un oggetto o una scena non banale.

In seguito era richiesto di utilizzare MeshLab per sperimentare alcune funzioni su mesh date.

5.1 Blender

È stata modellata una piccola stanza, seguendo alcuni tutorial su YouTube. Di seguito alcuni screenshot e un breve commento:

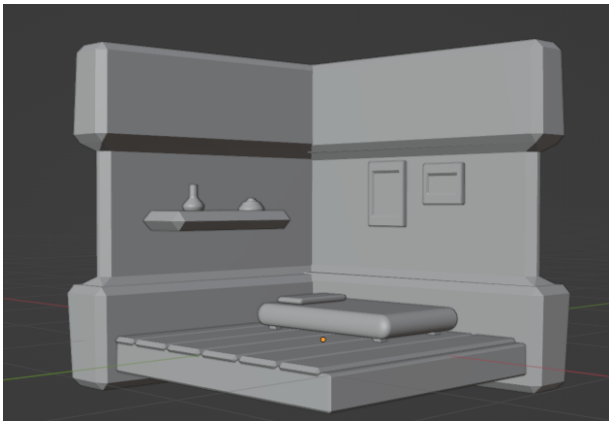

Figura 5.1: La stanza

La piccola stanza contiene un letto, due cornici e una mensola con sopra due vasi.

La base è costituita da un piano sopra al quale è stato creato un pavimento che desse l'idea di essere fatto di assi di legno.
Il pavimento è stato realizzato tramite un modificatore "Array" applicato ad un piano adeguatamente scalato e inserito sopra alla base.

Le pareti sono state create a partire da un cubo da cui sono state rimosse quattro facce.
Per realizzare le parti più spesse è stato applicato un taglio al centro delle pareti e poi sono state estruse lungo le normali le parti superiori e inferiori.
Infine è stato aggiunto un modificatore "Smussa" per ottenere un effetto migliore.

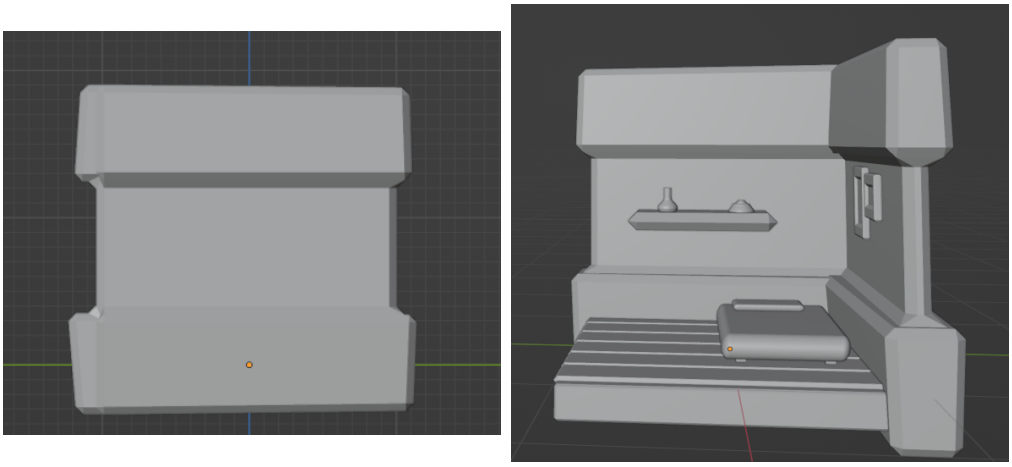

Figura 5.2: Punto di vista da asse x e -x

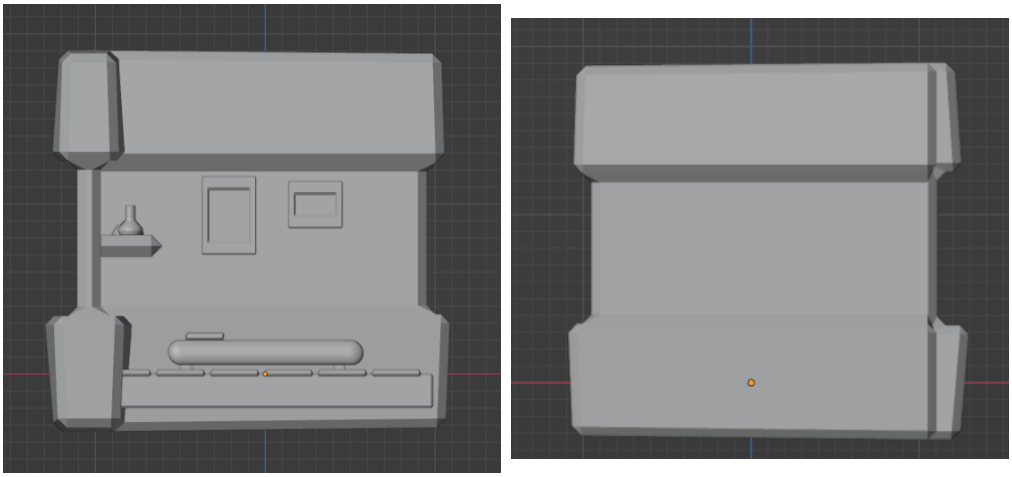

Figura 5.3: Punto di vista da asse y e -y

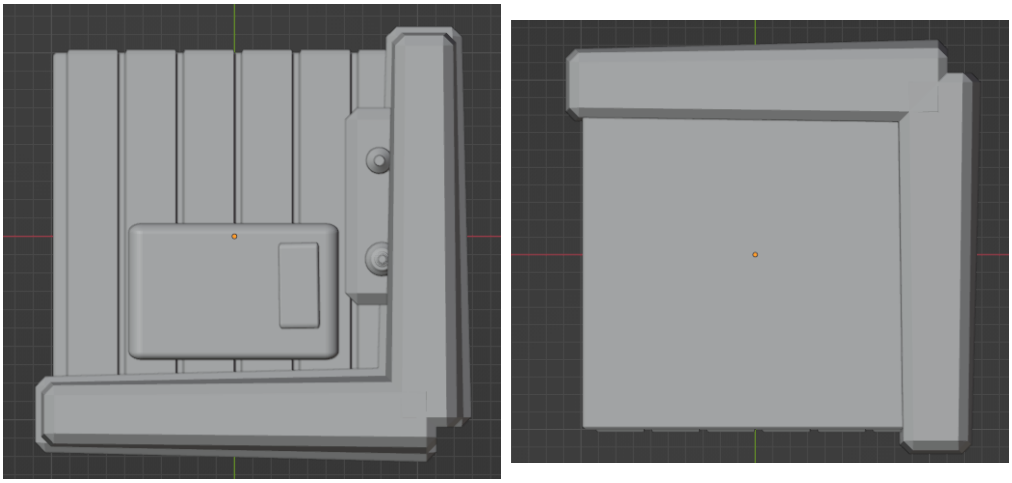

Figura 5.4: Punto di vista da asse z e -z

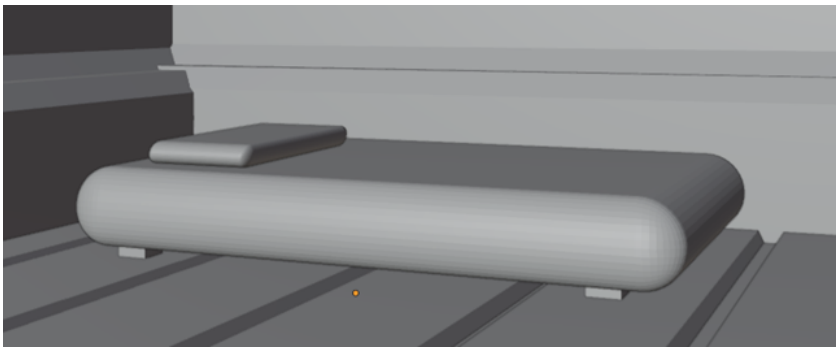

Figura 5.5: Dettaglio del letto

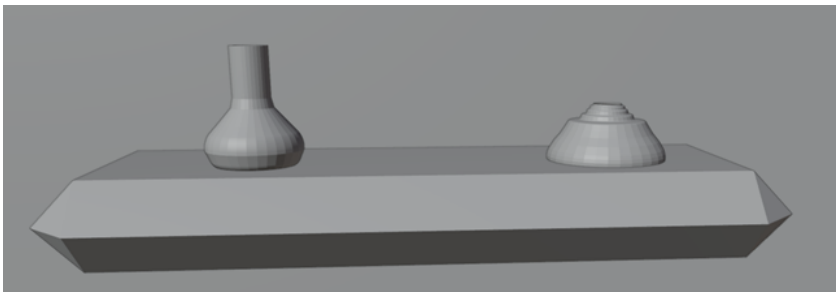

Figura 5.6: Dettaglio dei vasi

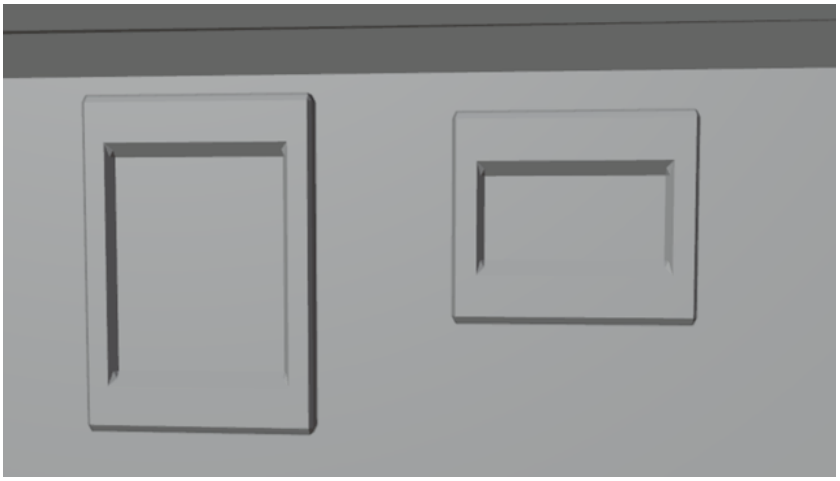

Figura 5.7: Dettaglio delle cornici

Le cornici sono state realizzate estrudendo due piani e incassandone la parte centrale mediante la funzione di inset. In seguito è stato aggiunto un modificatore "Smussa".
Similmente è stata realizzata la mensola.

I vasi sono stati creati a partire da due basi circolari estruse e scalate a piacere fino ad ottenere il risultato desiderato. Una volta creati i vasi è stata rimossa la faccia superiore in modo da creare un buco.

Infine, il letto e il cuscino sono due piani sovrapposti adeguatamente modellati. I piedini del letto sono quattro cubi.

5.2 Meshlab

Per questa seconda parte era richiesto di sperimentare le seguenti funzionalità di Meshlab:

1. Ricostruzione di oggetti mesh a partire da nuvole di punti;
2. Utilizzare i tool Fill Hole/Mesh Repair per la chiusura di una mesh parzialmente corrotta;
3. Applicare un filtro di denoising (fairing) ad una mesh perturbata;
4. Semplificare a più livelli una mesh con un numero elevato di elementi;
5. Utilizzare gli strumenti di misura della qualità della superficie (curvatura).

5.2.1 Ricostruzione mesh

Per il primo punto è stata ricostruita la mesh di un camaleonte partendo dalla relativa nuvola di punti mediante la funzione `Screened Poisson`.

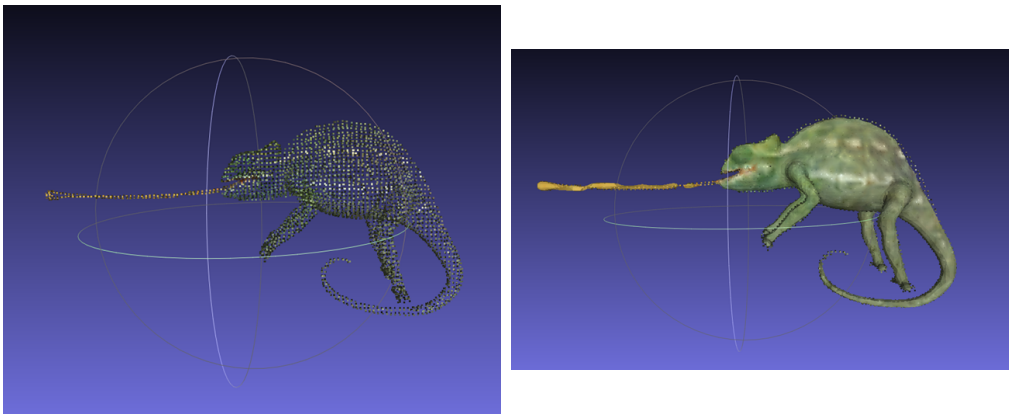

Figura 5.8: Nuvola di punti e camaleonte ricostruito

5.2.2 Chiusura di una mesh parzialmente corrotta

In questo caso era necessario chiudere una mesh parzialmente corrotta: è stata scelta la testa della dea Minerva che presentava un buco in prossimità del naso.
È stata riparata mediante la funzione `Close holes`.

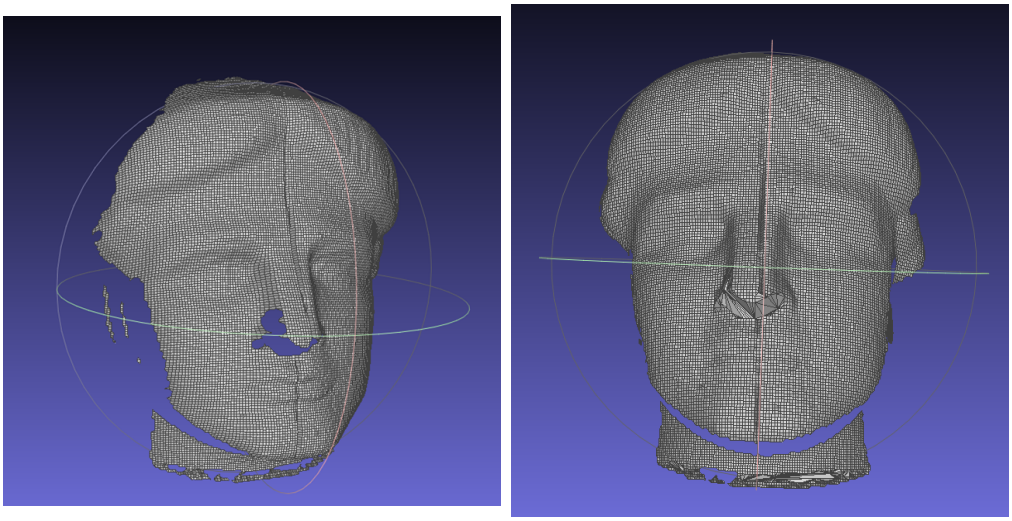

Figura 5.9: Testa di Minerva e relativa chiusura

5.2.3 Fairing

Per il punto 3 era necessario eliminare le perturbazioni dalla mesh. Per farlo è stato utilizzato lo strumento `Scale dependent Laplacian smooth`.

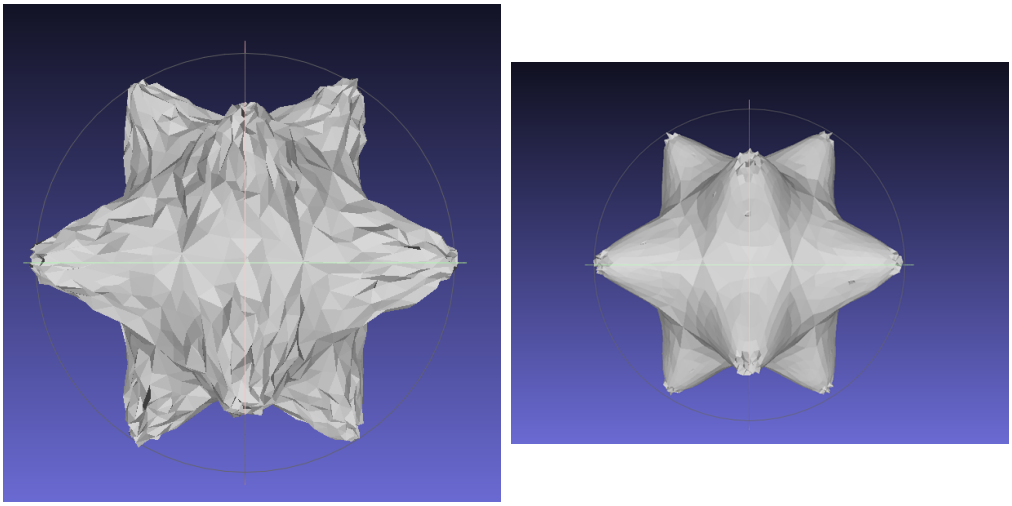

Figura 5.10: Eliminazione della perturbazione

5.2.4 Decimazione

Questo punto richiedeva di semplificare una mesh con numerosi elementi: per farlo è stato utilizzato il tool `Quadric edge collapse decimation` riducendo i vertici da più di 100000 a 1000.

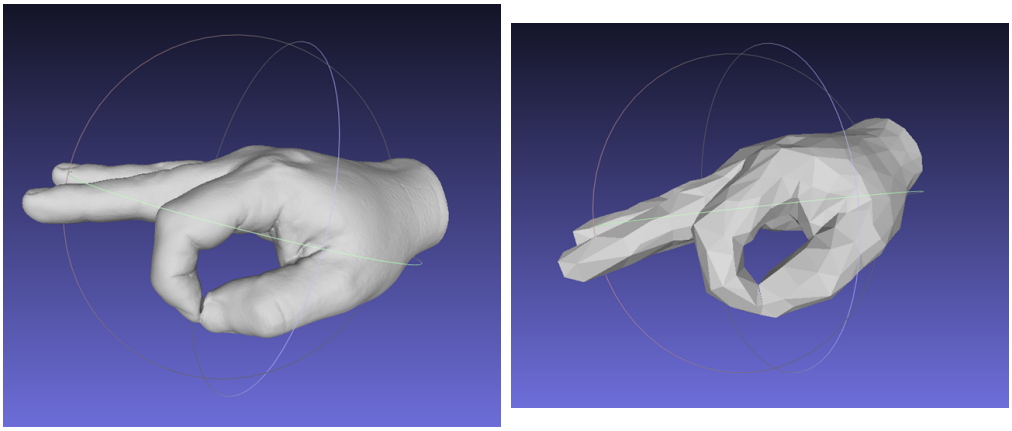

Figura 5.11: Decimazione della mano

5.2.5 Misura della qualità

Per l'ultimo punto è stato impiegato lo strumento `Compute curvatures principal directions` sulla stella da cui in precedenza era stata rimossa la perturbazione.

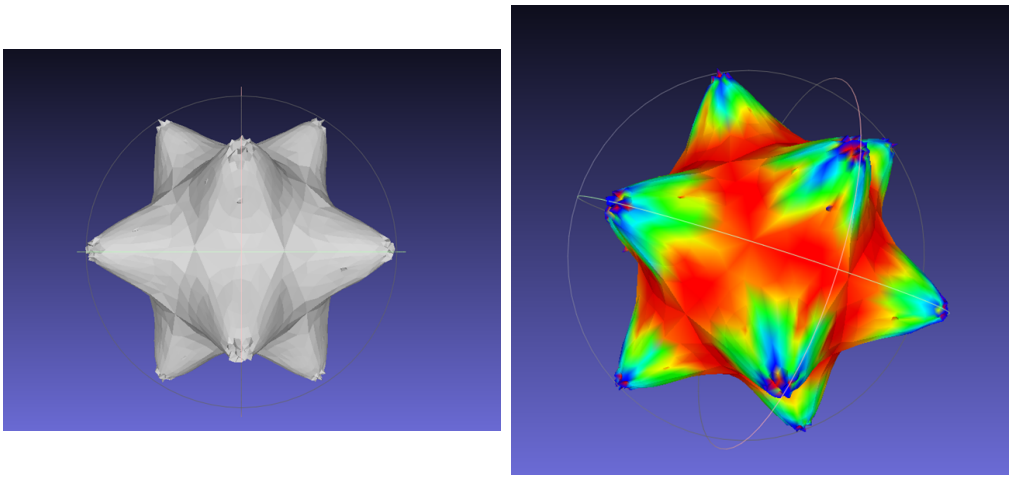

Figura 5.12: Misura della qualità della superficie