

PROGRAMMAZIONE GRAFICA ESERCIZI

1. Getting Started.....	4
4. Hello Window	4
4.4 Genera una finestra	4
4.6 Genera una finestra con colore di sfondo.....	4
5. Hello Triangle	5
5.5.2 Utilizza gli shader per disegnare un triangolo in finestra	5
5.6 Generare un rettangolo composto da due triangoli (visuale wireframe)	5
5.8 Esercizio 1: Disegna due triangoli vicini usando 'glDrawArrays' aggiungendo più vertici ai dati	5
5.8 Esercizio 2: Disegna due triangoli vicini usando due VAO e VBO diversi per i loro dati.	6
5.8 Esercizio 3: Disegna due triangoli, di cui uno di colore giallo utilizzando due programmi shader che si avvalgono di fragment shaders diversi.....	6
6. Shaders	7
6.3 Invio di valori dal vertex shader al fragment shader	7
6.4 Utilizzo degli uniforms per impostare attributi che possono cambiare nel tempo e per scambiare dati tra l'applicazione e gli shaders.....	7
6.5 Specificando tre colori come attributi ai vertici del triangolo, si sfrutta l'interpolazione tra frammenti del fragment shader per ottenere il seguente risultato	7
6.6 Utilizzo della classe shader e divisione in file di vertex shader e fragment shader	8
6.8 Esercizio 1: Modifica del vertex shader per capovolgere il triangolo	8
6.8 Esercizio 2: Utilizzo di un offset passato come parametro uniform per spostare il triangolo a destra dal vertex shader	8
6.8 Esercizio 3: Utilizzo della posizione del vertice come parametro RGB	9
7. Textures.....	10
7.0 Aggiunge la texture wall.jpg al triangolo	10
7.6 Aggiunge la texture container.jpg al rettangolo.....	10
7.6.1 Aggiunge la texture container.jpg al rettangolo ed interpola il colore con la posizione dei vertici	10
7.7 Mix di due texture con valore 50%/50%.....	11
7.8 Esercizio 1: Inverte la texture awesomeface.png	11
7.8 Esercizio 2: Replica solo la texture awesomeface.png (4 totali)	11
7.8 Esercizio 3: Utilizza solo la posizione centrale della texture come texture	12
7.8 Esercizio 4: Aumenta/riduce la percentuale di mix delle due texture con l'uso delle frecce da tastiera (freccia su per visualizzare awesomeface.png e freccia giù per visualizzare container.jpg)	12
8. Transformations.....	13
8.17 Scala e ruota il rettangolo	13
8.17 Esempio 2: Trasla il rettangolo in basso a destra e ruotalo nel tempo.....	13
8.19 Esercizio 1: Ruota nel tempo il rettangolo e traslalo in basso a destra.	13

8.19 Esercizio 2: Aggiungi a '8.17 Esempio 2' un rettangolo in alto a sinistra e scalalo nel tempo.....	14
9. Coordinate Systems	15
9.7 Sposta le coordinate dei vertici attraverso le matrici model, view e projection per ottenere un oggetto inclinato all'indietro, leggermente distante dall'osservatore e visualizzato con prospettiva.....	15
9.8 Disegna un cubo che ruota senza abilitare le informazioni di profondità	15
9.8.1 '9.8' con le informazioni di profondità abilitate	15
9.8.2 Disegna vari cubi in posizioni random con rotazione random.....	16
9.9 Esercizio 3: '9.8.2' ma ogni tre cubi uno ruota	16
10. Camera.....	17
10.2 Implementa la rotazione nel tempo della camera attorno alla scena	17
10.4 Implementa una camera che si muove alla stessa velocità su ogni sistema (deltaTime) (solo WASD su tastiera)	17
10.9 Implementa una camera che permette di muoversi liberamente in un ambiente 3D (WASD + scroll + posizione mouse) con una classe camera di tipo fly.....	17
10.9 Esercizio 1: La camera è di tipo FPS (mantiene l'osservatore a livello ground).....	18
10.9 Esercizio 2: Uguale agli esercizi precedenti ma viene creata la propria funzione LookAt	18
2. Lighting	19
12. Colors.....	19
12.1 Implementa un oggetto cubo ed un oggetto luce (cubo più piccolo).....	19
13. Basic Lighting	20
13.4 Utilizzo dell'illuminazione diffusa. Si noti la differenza di intensità illuminazione del cubo tanto maggiore è piccolo l'angolo tra il vettore normale e il vettore direzione della luce	20
13.6 Completa l'implementazione del sistema di illuminazione Phong nello spazio world.....	20
13.7 Esercizio 1: Utilizza seno e coseno per muovere la posizione della luce nel tempo	20
13.7 Esercizio 2: Implementa il Phong shading nello spazio view invece che world	21
13.7 Esercizio 3: Utilizzo del Gouraud shading al posto del Phong (meno preciso)	21
14. Materials.....	22
14.3 Stesso effetto raggiunto in precedenza ma con controllo completo sulla luce e sul materiale dell'oggetto	22
14.4 Esercizio 1: Simula il materiale plastica ciano con intensità luminosa impostata al massimo	22
15. Lighting Maps	23
15.1 Utilizzo di una diffuse map	23
15.3 Calcola la luce speculare dalla texture fornita.....	23
15.4 Esercizio 2: Inverte la luce speculare calcolata dalla texture.....	23
15.4 Esercizio 3: Utilizza una texture colorata per calcolare la luce speculare.....	24
15.4 Esercizio 4: Aggiunge una luce emissiva data da una texture (scritte verdi)	24
16. Light Casters.....	25
16.1 Creazione di una luce direzionale.....	25

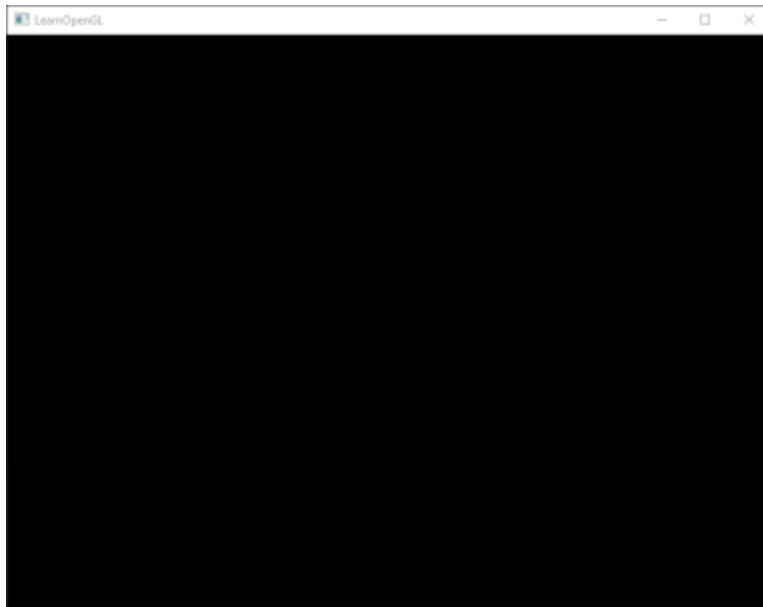
16.3.2 Creazione di una luce point con attenuazione	25
16.5 Creazione di una flashlight posizionata sulla camera	25
16.6 Creazione di una flashlight con bordi attenuati.....	26
17. Multiple lights.....	27
17.3 Crea una scena con luci multiple (ambientale, direzionale, spot e point).....	27
3. Model Loading.....	28
21. Stencil testing	28
21.3 Importa un modello (.obj) completo di texture attraverso Assimp (in particolare uno zaino). Creazione di classi Mesh e Model dedicate.....	28
5. Advanced Lighting.....	29
37.2.2 Utilizza il normal mapping per generare dettagli a basso costo attraverso la modifica di vettori delle normali per frammento senza dover modificare l'equazione dell'illuminazione	29
6. Esercitazione 1.....	30
7. Esercitazione 2.....	31

1. Getting Started

4. Hello Window

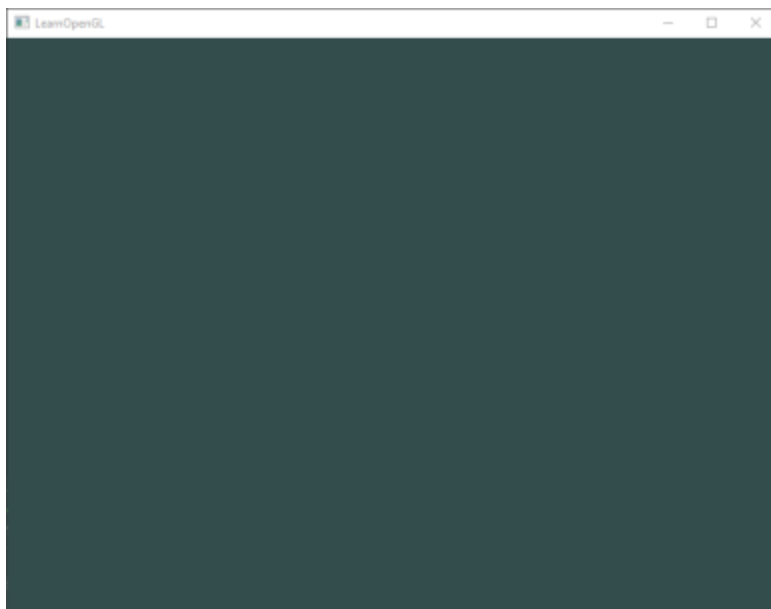
4.4 Genera una finestra

File [1](#) [4.4](#)



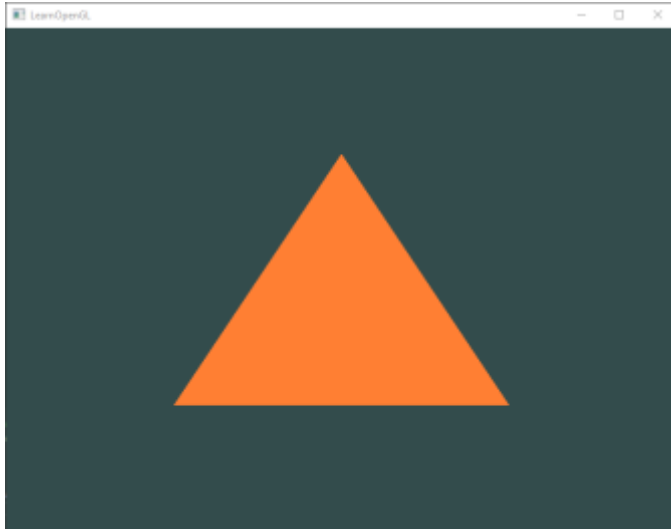
4.6 Genera una finestra con colore di sfondo

File [1](#) [4.6](#)



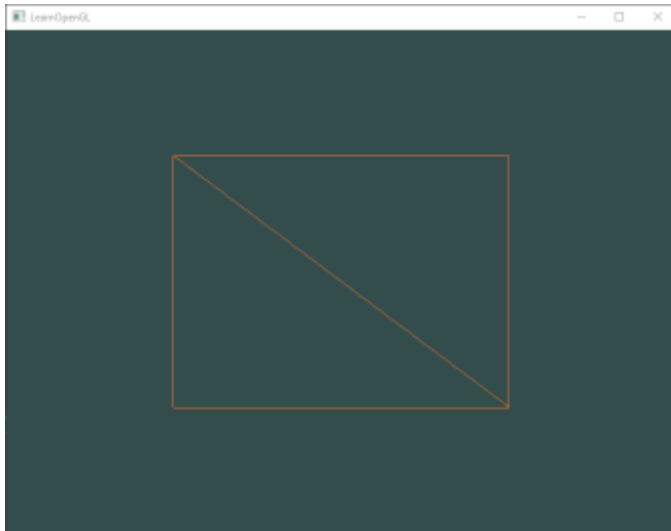
5. Hello Triangle

5.5.2 Utilizza gli shader per disegnare un triangolo in finestra



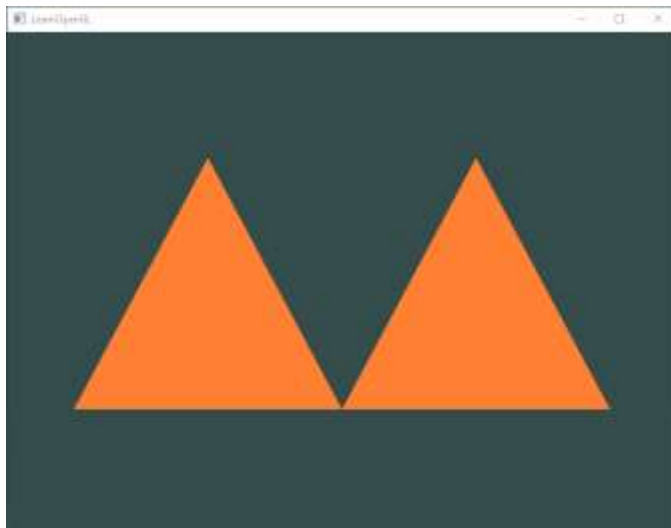
File [1_5.5.2](#)

5.6 Generare un rettangolo composto da due triangoli (visuale wireframe)



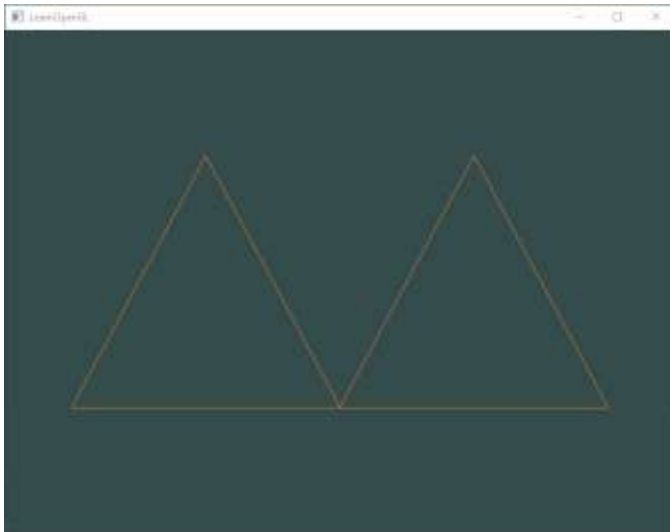
File [1_5.6](#)

5.8 Esercizio 1: Disegna due triangoli vicini usando 'glDrawArrays' aggiungendo più vertici ai dati



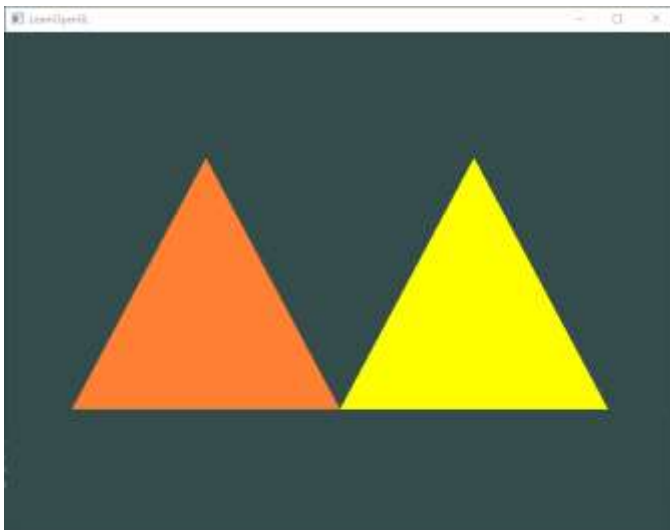
File [1_5.8_1](#)

5.8 Esercizio 2: Disegna due triangoli vicini usando due VAO e VBO diversi per i loro dati.



File [1_5.8_2](#)

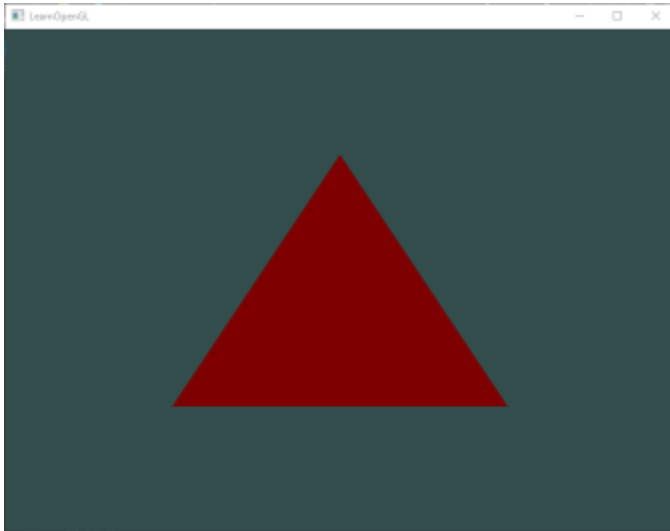
5.8 Esercizio 3: Disegna due triangoli, di cui uno di colore giallo utilizzando due programmi shader che si avvalgono di fragment shaders diversi



File [1_5.8_3](#)

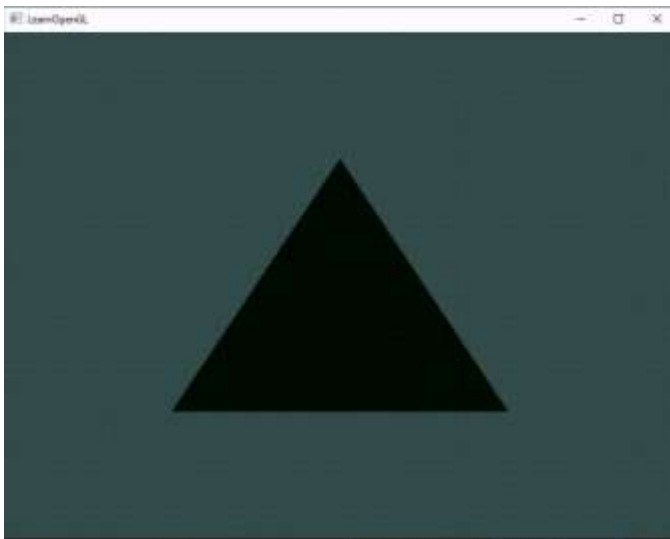
6. Shaders

6.3 Invio di valori dal vertex shader al fragment shader



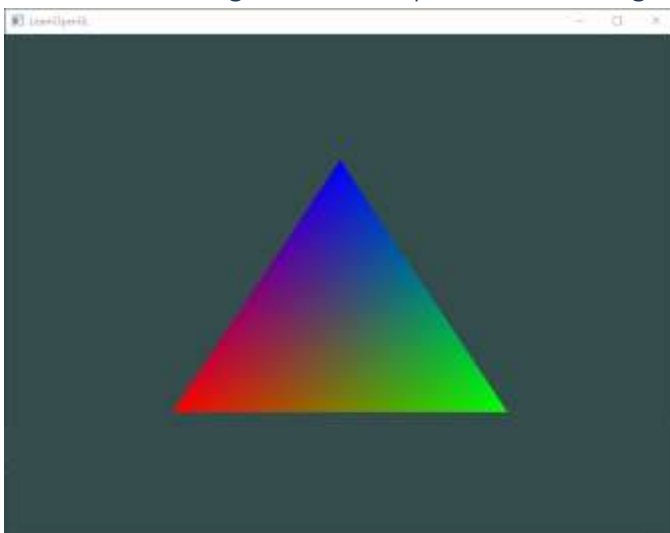
File [1_6.3](#)

6.4 Utilizzo degli uniforms per impostare attributi che possono cambiare nel tempo e per scambiare dati tra l'applicazione e gli shaders



File [1_6.4](#)

6.5 Specificando tre colori come attributi ai vertici del triangolo, si sfrutta l'interpolazione tra frammenti del fragment shader per ottenere il seguente risultato



File [1_6.5](#)

6.6 Utilizzo della classe shader e divisione in file di vertex shader e fragment shader



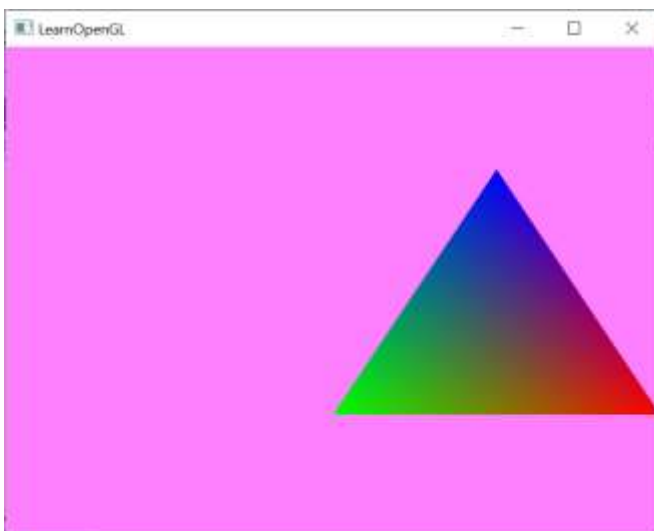
File [1](#) [6.6](#)

6.8 Esercizio 1: Modifica del vertex shader per capovolgere il triangolo



File [1](#) [6.8](#) [1](#)

6.8 Esercizio 2: Utilizzo di un offset passato come parametro uniform per spostare il triangolo a destra dal vertex shader



File [1](#) [6.8](#) [2](#)

6.8 Esercizio 3: Utilizzo della posizione del vertice come parametro RGB



File [1](#) [6.8](#) [3](#)

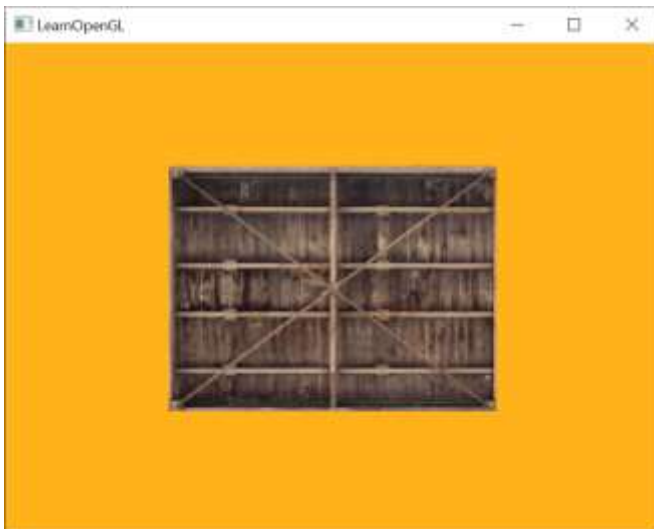
7. Textures

7.0 Aggiunge la texture wall.jpg al triangolo



File [1 7.0](#)

7.6 Aggiunge la texture container.jpg al rettangolo



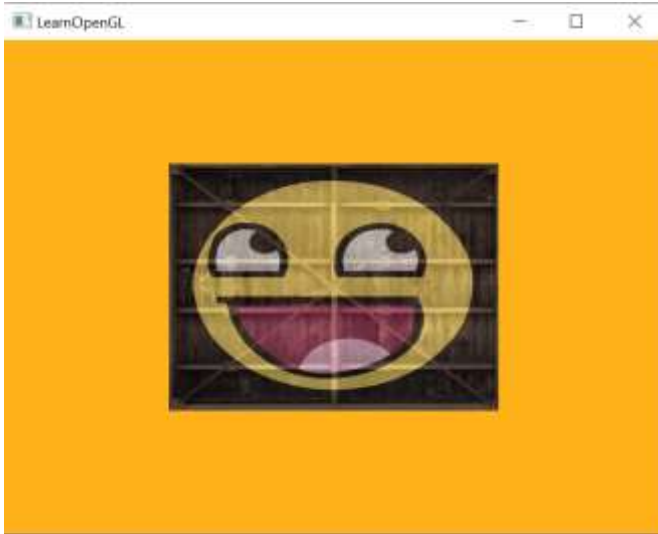
File [1 7.6](#)

7.6.1 Aggiunge la texture container.jpg al rettangolo ed interpola il colore con la posizione dei vertici



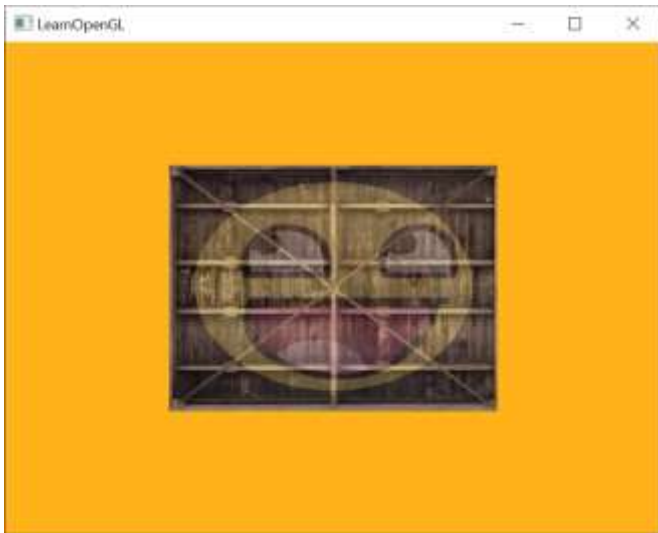
File [1 7.6.1](#)

7.7 Mix di due texture con valore 50%/50%



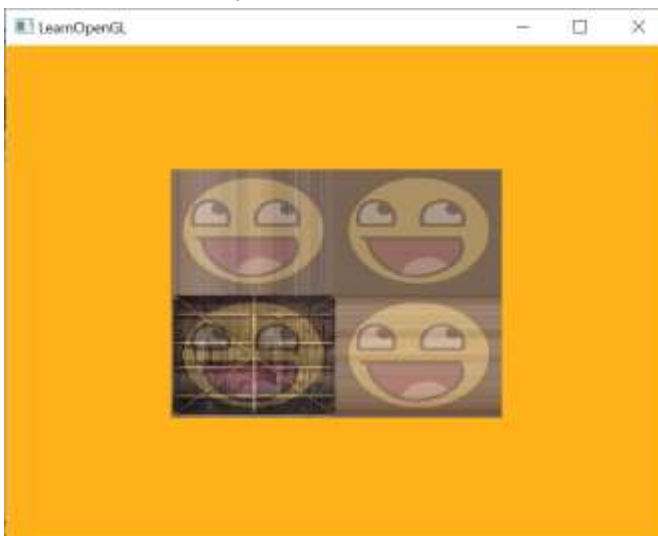
File [1](#) [7.7](#)

7.8 Esercizio 1: Inverte la texture awesomeface.png



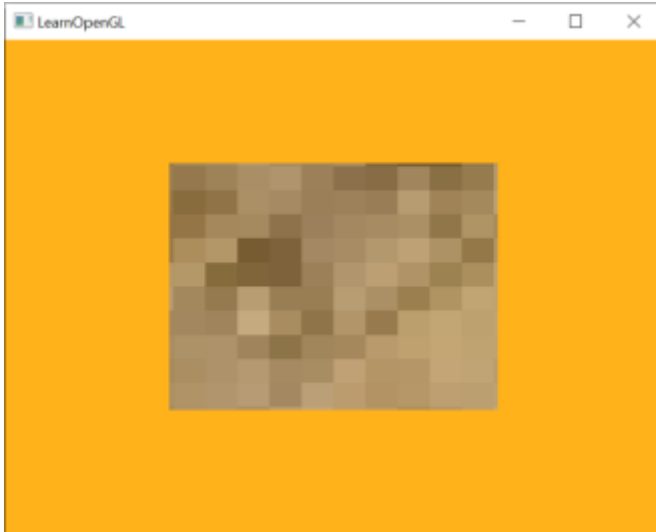
File [1](#) [7.8](#) [1](#)

7.8 Esercizio 2: Replica solo la texture awesomeface.png (4 totali)



File [1](#) [7.8](#) [2](#)

7.8 Esercizio 3: Utilizza solo la posizione centrale della texture come texture



File [1 7.8 3](#)

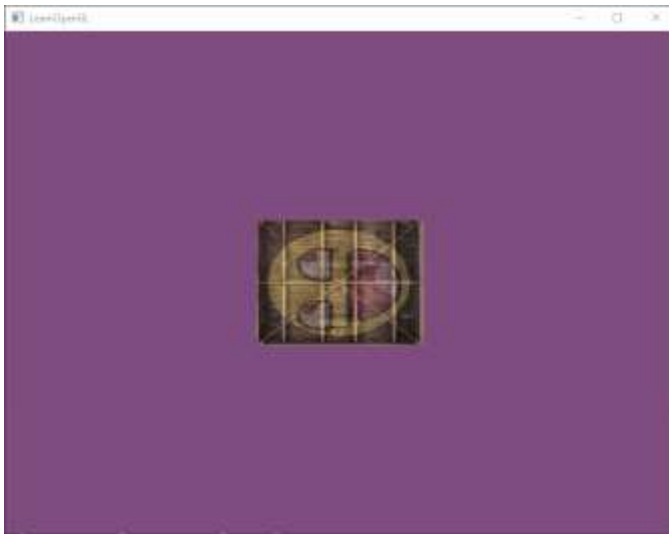
7.8 Esercizio 4: Aumenta/riduce la percentuale di mix delle due texture con l'uso delle frecce da tastiera (freccia su per visualizzare awesomeface.png e freccia giù per visualizzare container.jpg)



File [1 7.8 4](#)

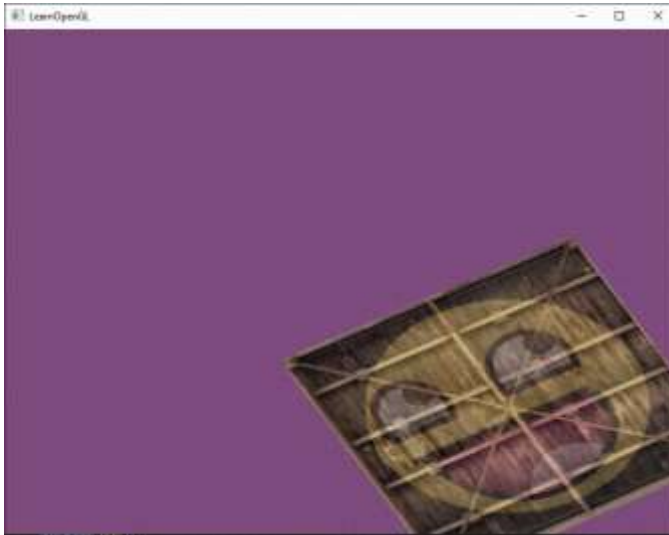
8. Transformations

8.17 Scala e ruota il rettangolo



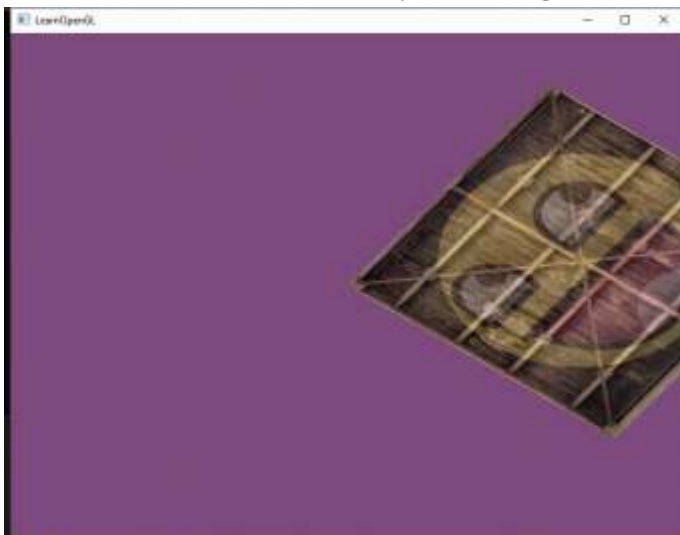
File [1 8.17](#)

8.17 Esempio 2: Trasla il rettangolo in basso a destra e ruotalo nel tempo



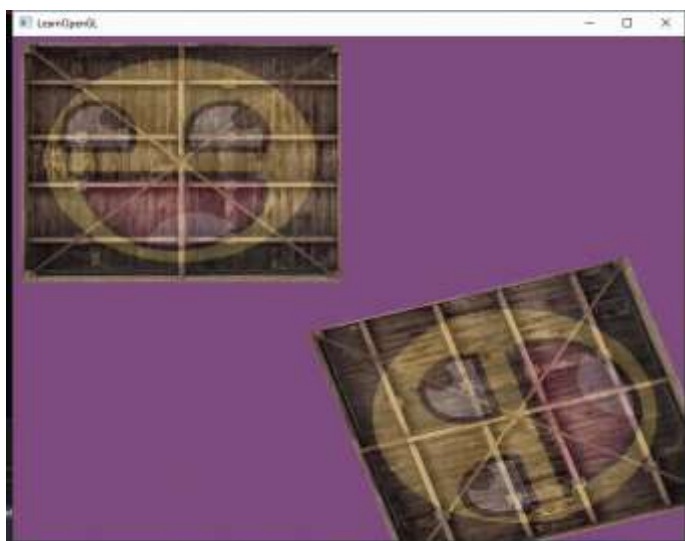
File [1 8.17 2](#)

8.19 Esercizio 1: Ruota nel tempo il rettangolo e traslalo in basso a destra.



File [1 8.19 1](#)

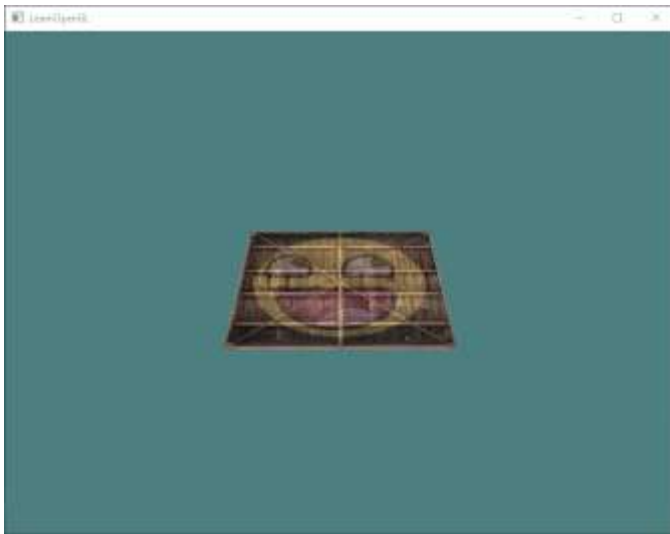
8.19 Esercizio 2: Aggiungi a '8.17 Esempio 2' un rettangolo in alto a sinistra e scalalo nel tempo



File [1](#) [8.19](#) [2](#)

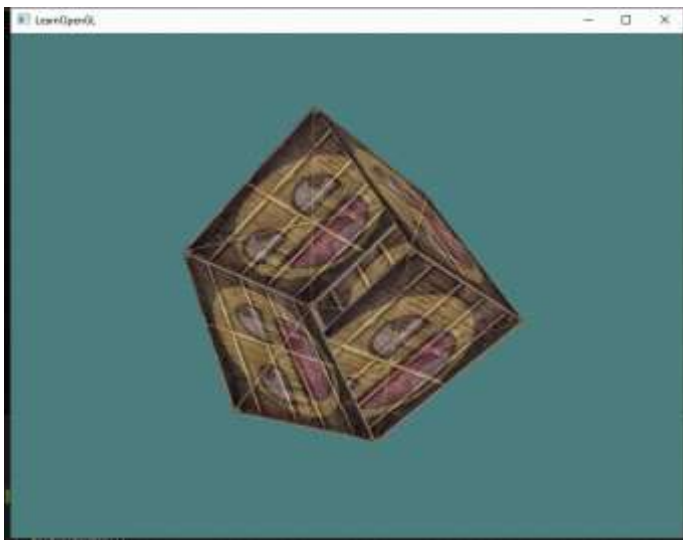
9. Coordinate Systems

9.7 Sposta le coordinate dei vertici attraverso le matrici model, view e projection per ottenere un oggetto inclinato all'indietro, leggermente distante dall'osservatore e visualizzato con prospettiva



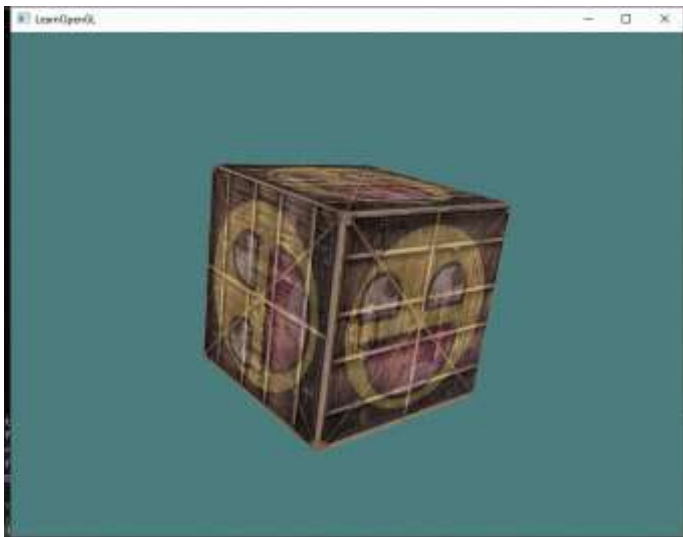
File [1_9.7](#)

9.8 Disegna un cubo che ruota senza abilitare le informazioni di profondità



File [1_9.8](#)

9.8.1 '9.8' con le informazioni di profondità abilitate



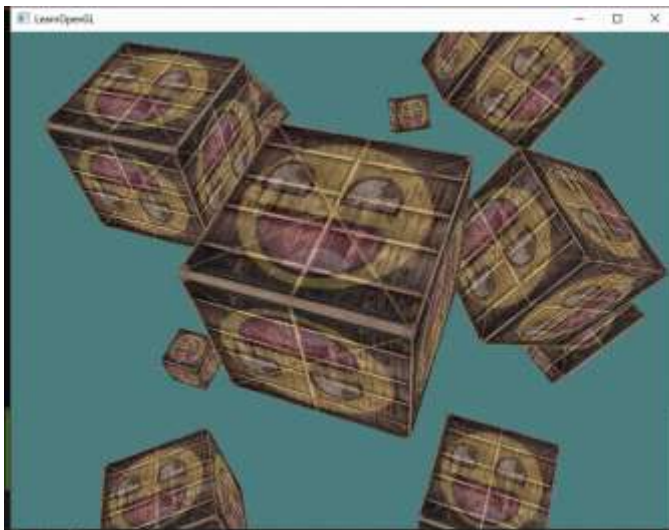
File [1_9.8.1](#)

9.8.2 Disegna vari cubi in posizioni random con rotazione random



File [1_9.8.2](#)

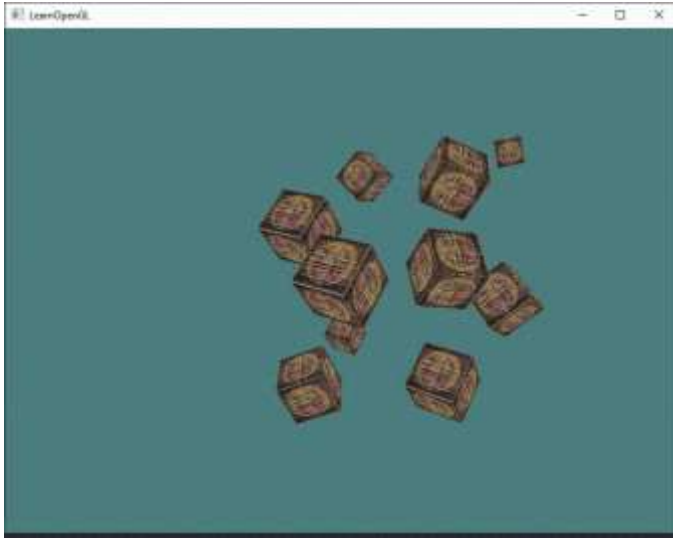
9.9 Esercizio 3: '9.8.2' ma ogni tre cubi uno ruota



File [1_9.9_3](#)

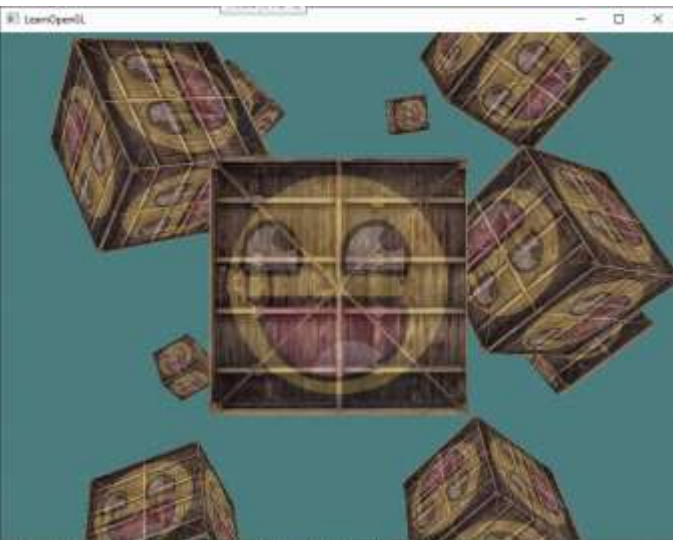
10. Camera

10.2 Implementa la rotazione nel tempo della camera attorno alla scena



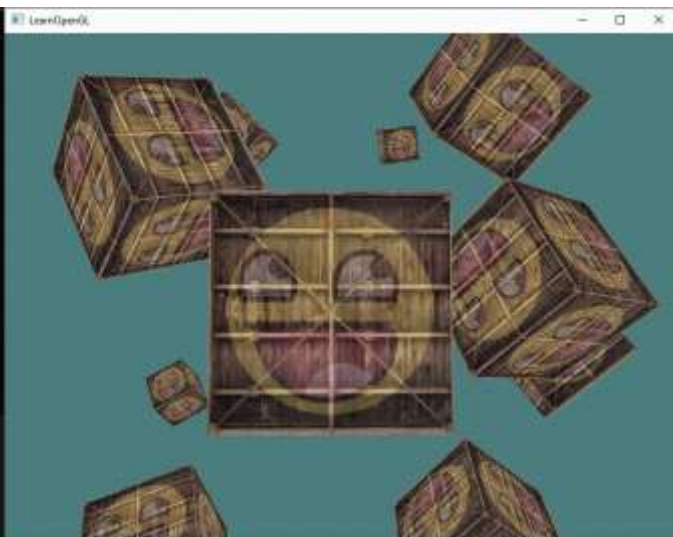
File [1_10.2](#)

10.4 Implementa una camera che si muove alla stessa velocità su ogni sistema (deltaTime) (solo WASD su tastiera)



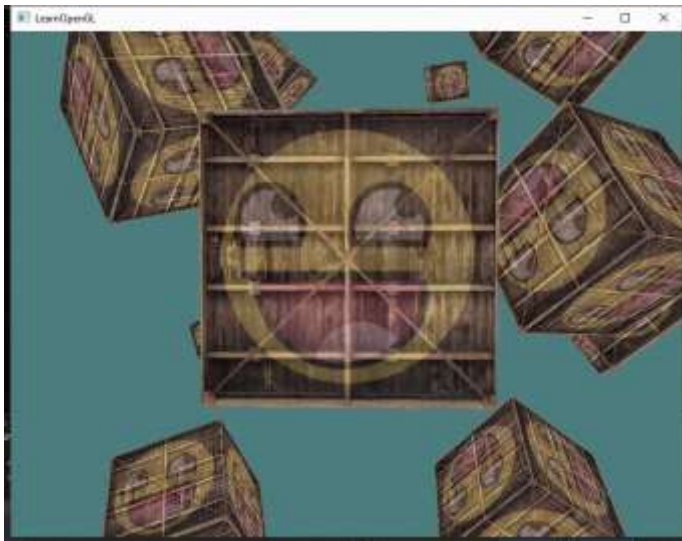
File [1_10.4](#)

10.9 Implementa una camera che permette di muoversi liberamente in un ambiente 3D (WASD + scroll + posizione mouse) con una classe camera di tipo fly



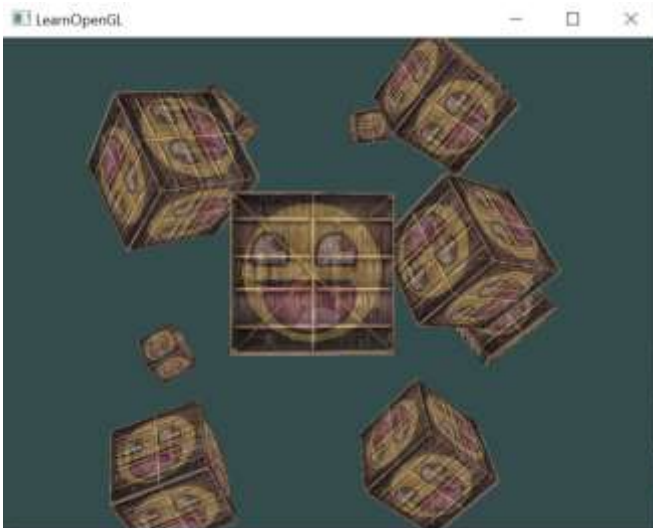
File [1_10.9](#) (+ camera.h)

10.9 Esercizio 1: La camera è di tipo FPS (mantiene l'osservatore a livello ground)



File [1 10.9 1](#) (camera.h con ground level (xz plane))

10.9 Esercizio 2: Uguale agli esercizi precedenti ma viene creata la propria funzione LookAt

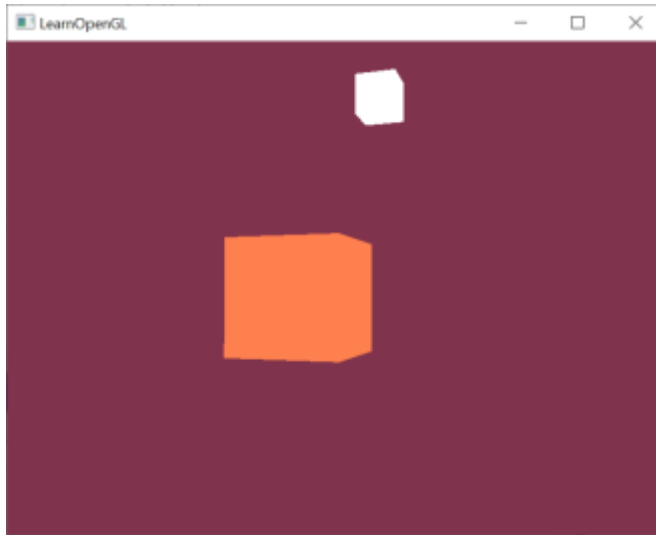


File [1 10.9 2](#)

2. Lighting

12. Colors

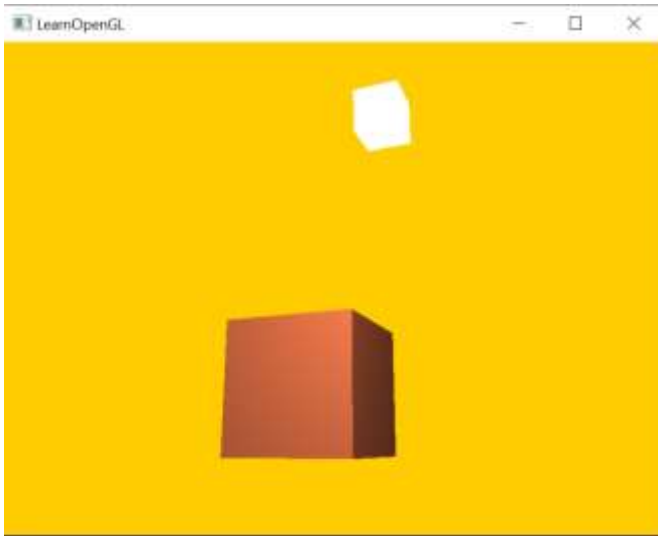
12.1 Implementa un oggetto cubo ed un oggetto luce (cubo più piccolo)



File [2_12.1](#)

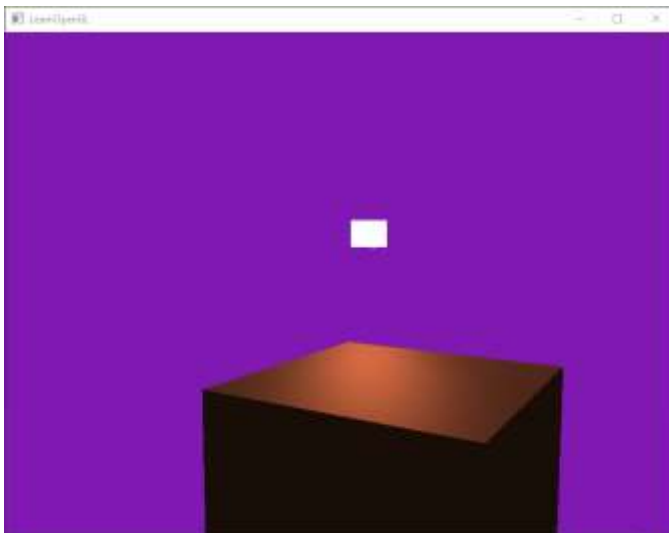
13. Basic Lighting

13.4 Utilizzo dell'illuminazione diffusa. Si noti la differenza di intensità illuminazione del cubo tanto maggiore è piccolo l'angolo tra il vettore normale e il vettore direzione della luce



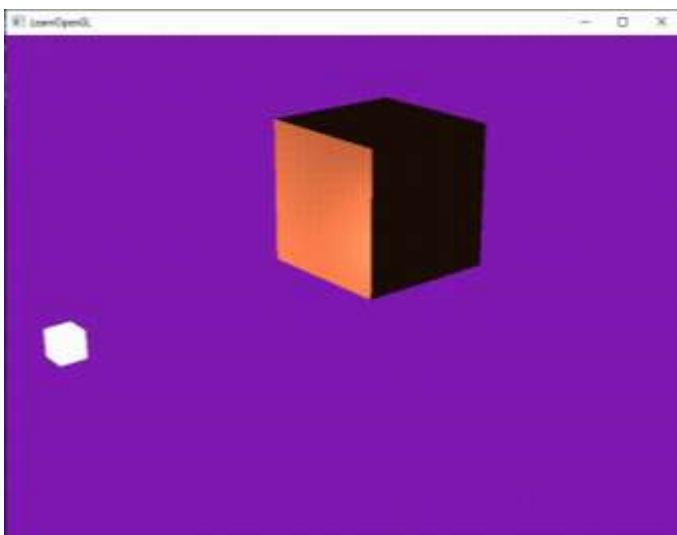
File [2_13.4](#)

13.6 Completa l'implementazione del sistema di illuminazione Phong nello spazio world



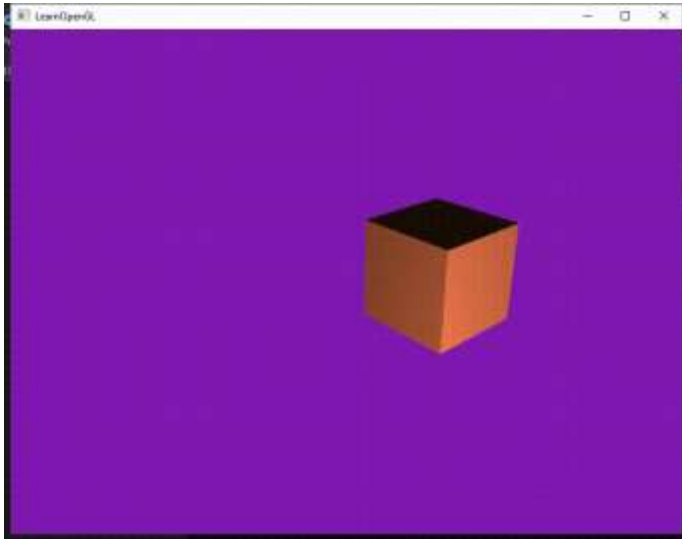
File [2_13.6](#)

13. 7 Esercizio 1: Utilizza seno e coseno per muovere la posizione della luce nel tempo



File [2_13.7_1](#)

13.7 Esercizio 2: Implementa il Phong shading nello spazio view invece che world



File [2_13.7_2](#) (usare AWDS tastiera)

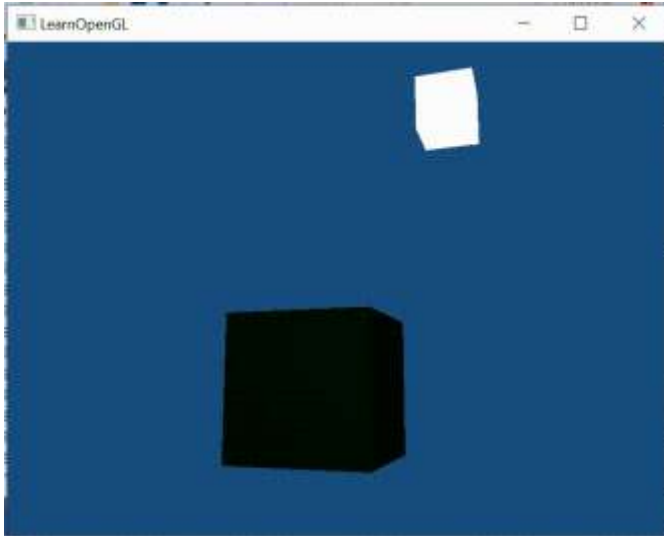
13.7 Esercizio 3: Utilizzo del Gouraud shading al posto del Phong (meno preciso)



File [2_13.7_3](#)

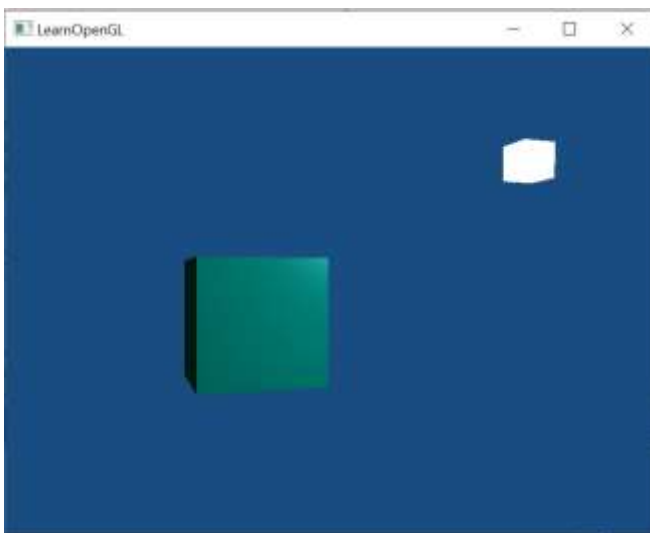
14. Materials

14.3 Stesso effetto raggiunto in precedenza ma con controllo completo sulla luce e sul materiale dell'oggetto



File [2_14.3](#)

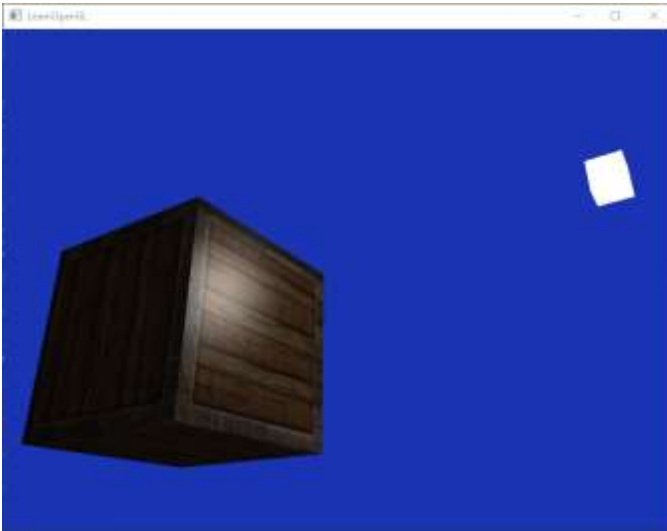
14.4 Esercizio 1: Simula il materiale plastica ciano con intensità luminosa impostata al massimo



File [2_14.4_1](#)

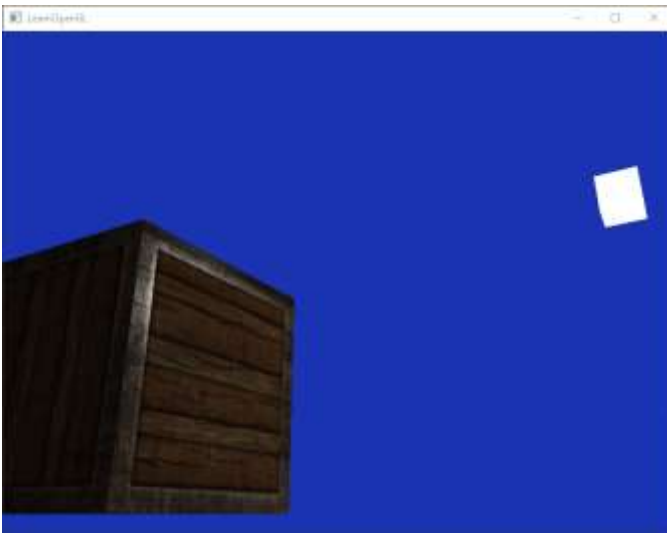
15. Lighting Maps

15.1 Utilizzo di una diffuse map



File [2_15.1](#)

15.3 Calcola la luce speculare dalla texture fornita



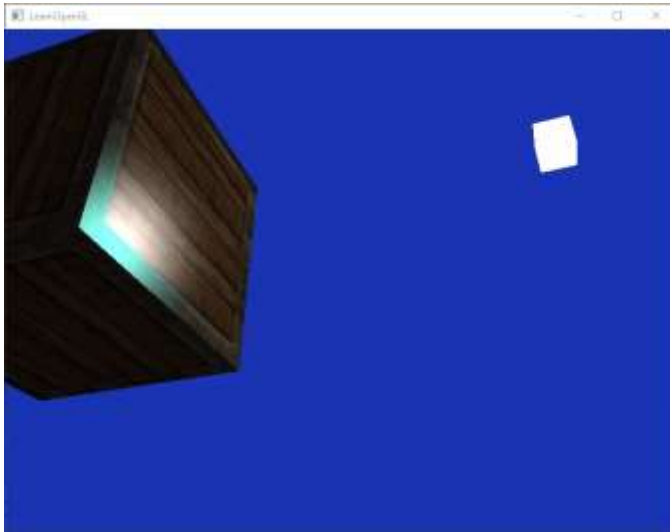
File [2_15.3](#)

15.4 Esercizio 2: Inverte la luce speculare calcolata dalla texture



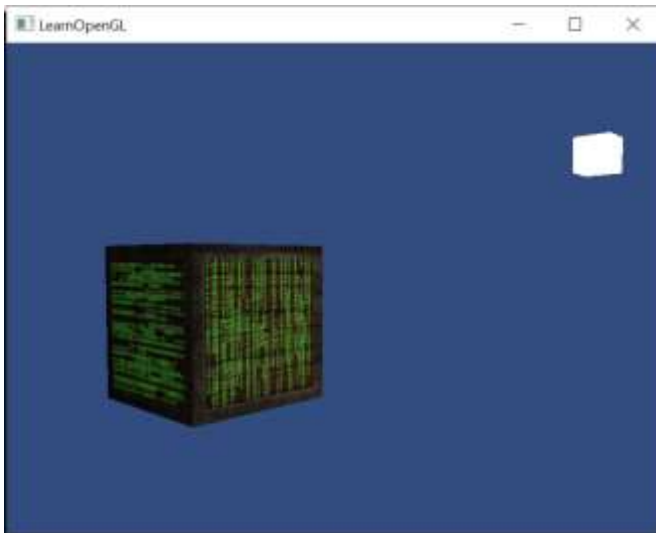
File [2_15.4_1](#)

15.4 Esercizio 3: Utilizza una texture colorata per calcolare la luce speculare



File [2_15.4_2](#)

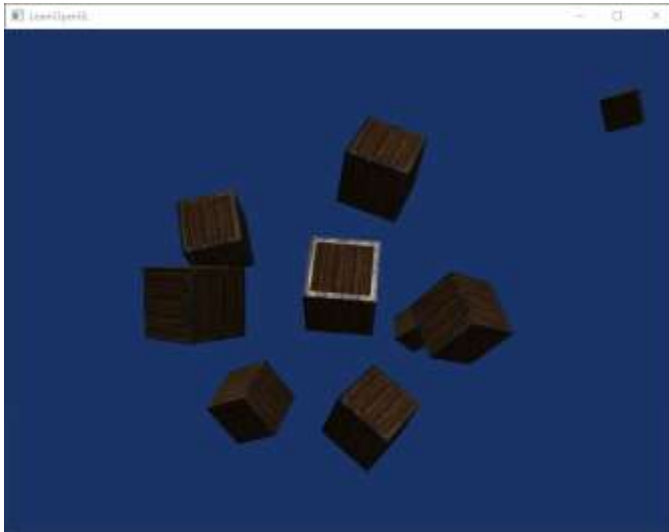
15.4 Esercizio 4: Aggiunge una luce emissiva data da una texture (scritte verdi)



File [2_15.4_3](#)

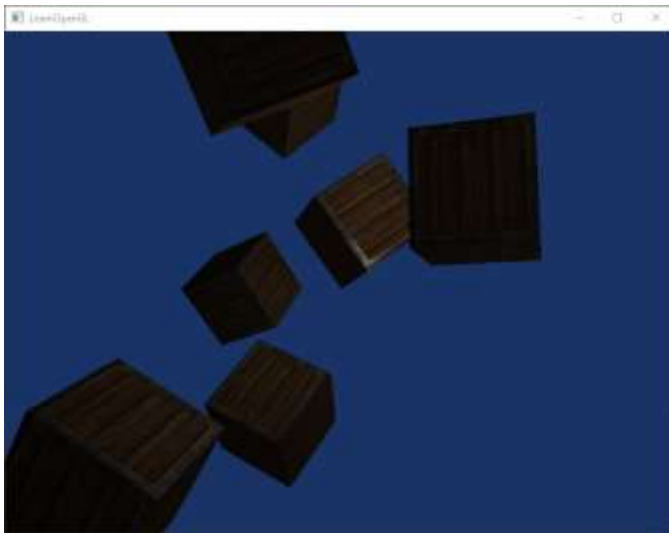
16. Light Casters

16.1 Creazione di una luce direzionale



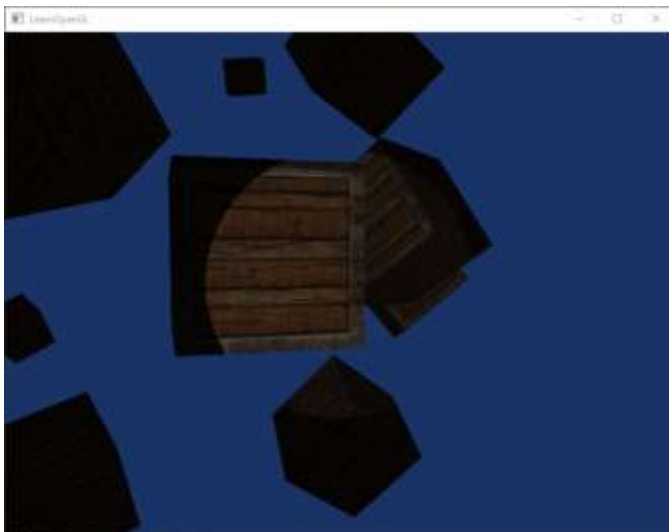
File [2_16.1](#)

16.3.2 Creazione di una luce point con attenuazione



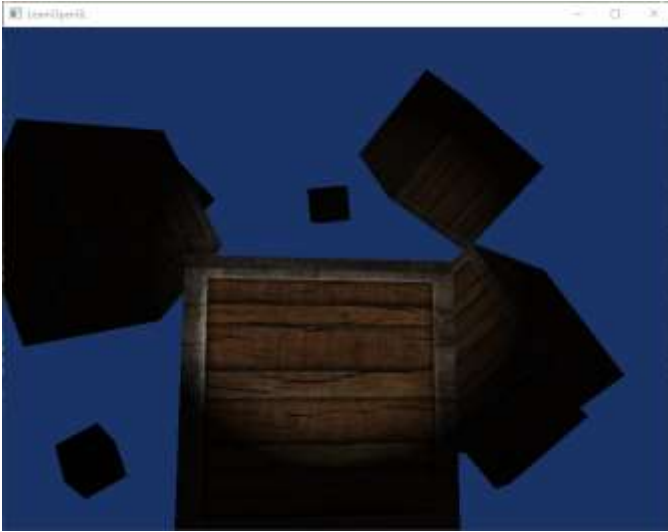
File [2_16.3.2](#)

16.5 Creazione di una flashlight posizionata sulla camera



File [2_16.5](#)

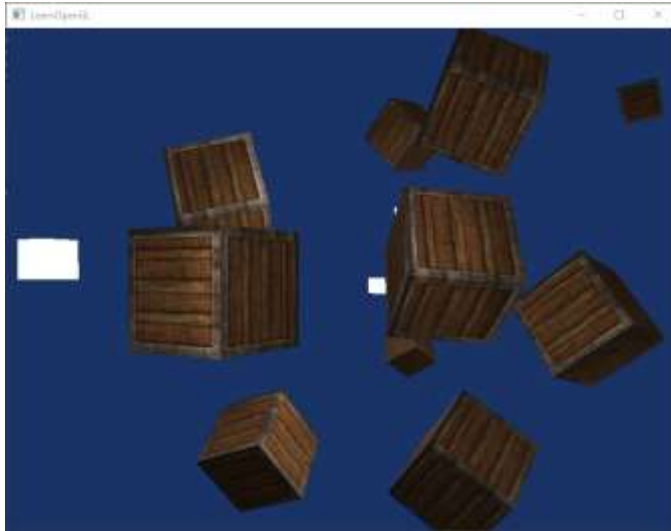
16.6 Creazione di una flashlight con bordi attenuati



File [2_16.6](#)

17. Multiple lights

17.3 Crea una scena con luci multiple (ambientale, direzionale, spot e point)

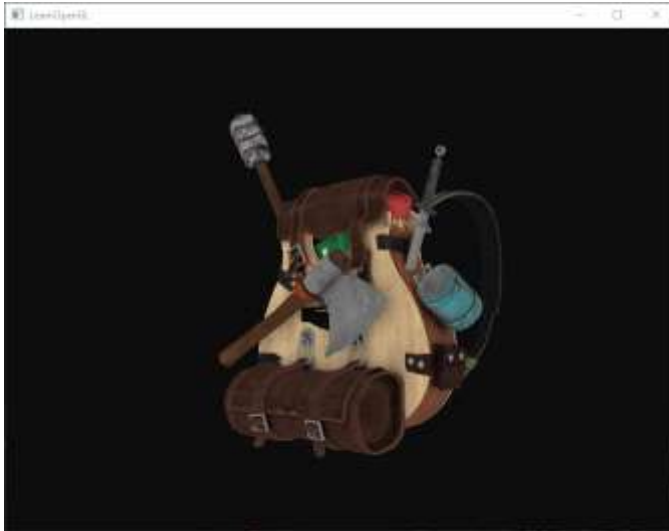


File [2_17.3](#)

3. Model Loading

21. Stencil testing

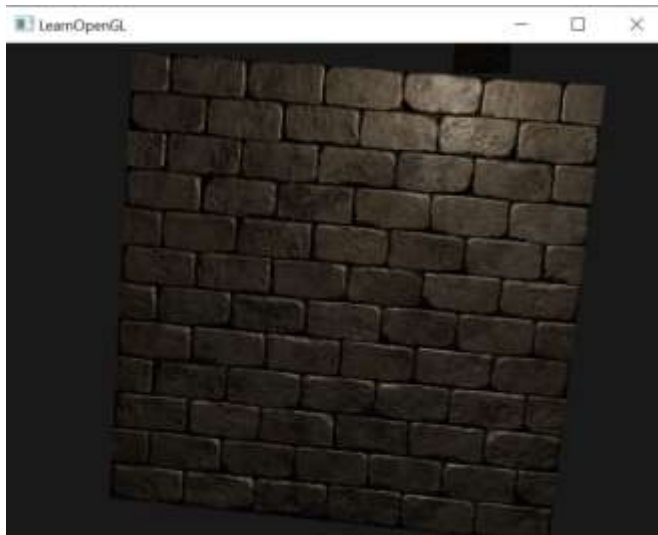
21.3 Importa un modello (.obj) completo di texture attraverso Assimp (in particolare uno zaino).
Creazione di classi Mesh e Model dedicate



File [3 21.3](#)

5. Advanced Lighting

37.2.2 Utilizza il normal mapping per generare dettagli a basso costo attraverso la modifica di vettori delle normali per frammento senza dover modificare l'equazione dell'illuminazione



File [5_37.2.2](#)

6. Esercitazione 1

L'esercitazione 1 comprende tutto quello visto fin'ora.

Viene istanziato un modello statico, il "backpack.obj" utilizzato già nel capitolo del Model Loading (3_21.3).

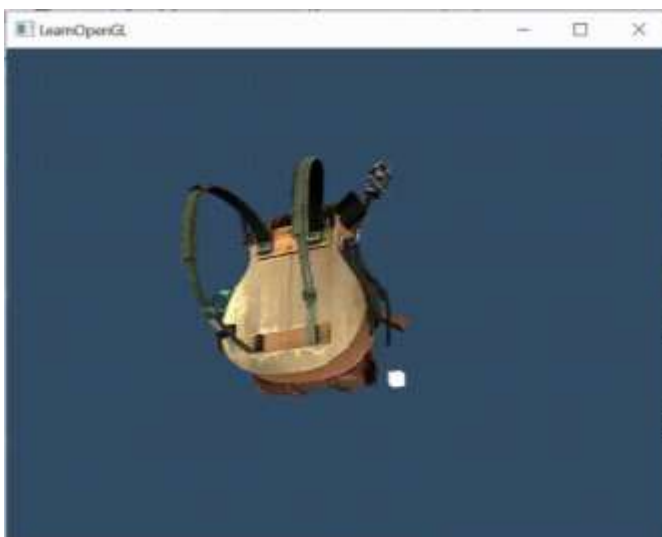
Al modello statico viene aggiunto il lighting che è formato da:

- 1 luce direzionale
- 4 point light, di cui una ruota attorno al modello
- 1 spotlight posta sulla camera.

L'illuminazione passa quindi attraverso la definizione di tutte le strutture dati necessarie che poi dal programma vengono passati allo shader dedicato.

Il lighting viene calcolato all'interno del tangent space.

Sono inoltre implementate le solite funzioni riguardo l'input utente, in particolare il movimento della camera con i tasti WASD e la direzione di essa decisa dal movimento del mouse.



7. Esercitazione 2

Le POINT LIGHTS sono una tecnica usata per generare ombre dinamiche in tutte le direzioni, in quanto nel mondo che ci circonda una luce proveniente da un punto emette ombre in tutte le direzioni.

La tecnica è per lo più simile alla mappatura direzionale delle ombre: si genera una mappa di profondità dalla prospettiva della luce, si campiona la mappa di profondità in base alla posizione corrente del frammento e si confronta ogni frammento con il valore di profondità memorizzato per vedere se è in ombra.

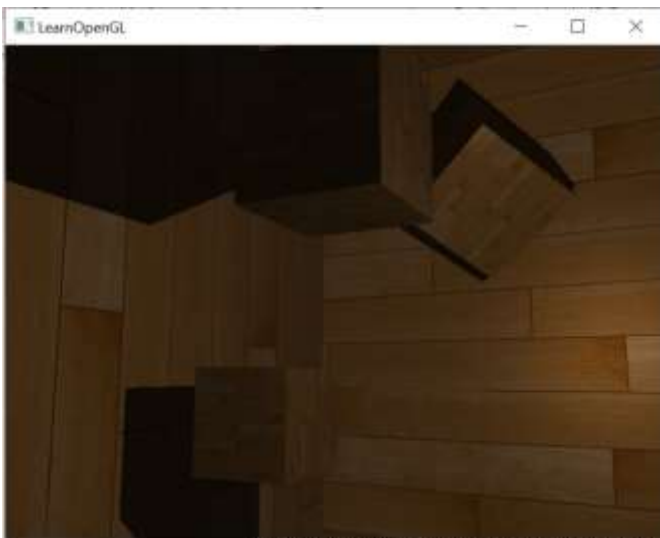
La differenza principale tra la mappatura direzionale delle ombre e la mappatura omnidirezionale delle ombre è la mappa di profondità utilizzata (Depth Map).

La mappa di profondità di cui abbiamo bisogno richiede il rendering della scena da tutte le direzioni circostanti di un punto luce e quindi una normale mappa di profondità 2D non funziona. Si potrebbe perciò usare una cubemap. Con essa è possibile eseguire il rendering dell'intera scena su ciascuna delle facce di una cubemap e campionare questi dati come valori di profondità circostanti del punto luce, poiché una cubemap può memorizzare i dati completi dell'ambiente con solo 6 facce.

La cubemap di profondità (Depth Cubemap) generata viene quindi passata allo shader del frammento di illuminazione che campiona la cubemap con un vettore di direzione per ottenere la profondità più vicina (dalla prospettiva della luce) a quel punto.

La cubemap può essere piuttosto costosa, poiché sono necessarie molte chiamate di rendering per questa singola mappa di profondità.

Perciò viene utilizzato un approccio alternativo, utilizzando un piccolo trucco nello shader geometrico che ci permette di costruire la cubemap di profondità con un solo passaggio di rendering.

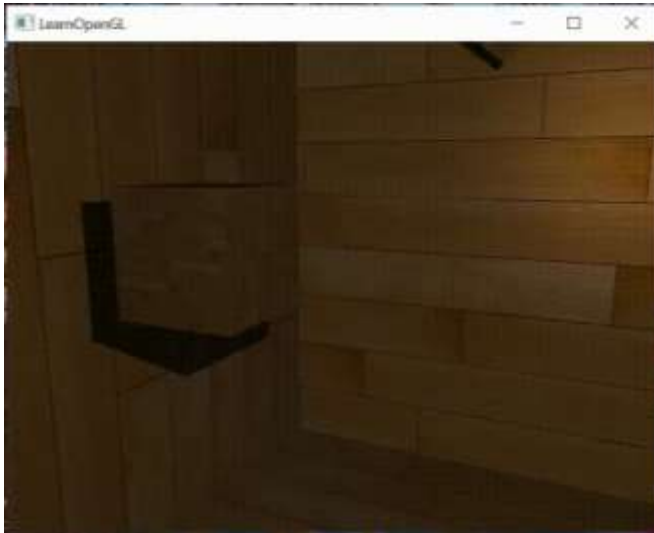


Per prima cosa, è necessario creare un cubemap e assegnare a ciascuna delle facce del cubemap un'immagine texture 2D con valore di profondità. Normalmente si attacca una singola faccia di una texture Cubemap all'oggetto framebuffer e si renderizza la scena 6 volte. Dal momento che utilizzeremo uno shader geometrico, che ci consente di eseguire il rendering di tutte le facce in un unico passaggio, possiamo collegare direttamente la cubemap come allegato di profondità del framebuffer con `glFramebufferTexture`.

Con le mappe d'ombra omnidirezionali abbiamo due passaggi di rendering: prima si genera la Cubemap di profondità e poi si utilizza la mappa di profondità nel normale passaggio di rendering per aggiungere le ombre alla scena.

Il processo è esattamente lo stesso della mappatura delle ombre direzionale, anche se questa volta renderizziamo e utilizziamo una texture di profondità Cubemap anziché una texture di profondità 2D.

Con il framebuffer e la Cubemap impostati, si ha bisogno di un modo per trasformare tutta la geometria della scena in spazi di luce in tutte e 6 le direzioni della luce. Si ha bisogno dunque di una matrice di trasformazione dello spazio di luce T, una per ogni faccia.



Ogni matrice di trasformazione dello spazio di luce contiene sia una matrice di proiezione che una matrice di vista. Per la matrice di proiezione utilizzeremo una matrice di proiezione prospettica; la sorgente di luce rappresenta un punto nello spazio, quindi la proiezione prospettica è la più sensata. Ogni matrice di trasformazione dello spazio luminoso utilizza la stessa matrice di proiezione: è importante notare che il parametro del campo visivo di `glm::perspective` è impostato a 90 gradi. Impostando questo parametro a 90 gradi, ci assicuriamo che il campo visivo sia esattamente

grande abbastanza per riempire una singola faccia della Cubemap, in modo che tutte le facce si allineino correttamente tra loro ai bordi. Poiché la matrice di proiezione non cambia per ogni direzione, possiamo riutilizzarla per ciascuna delle 6 matrici di trasformazione.

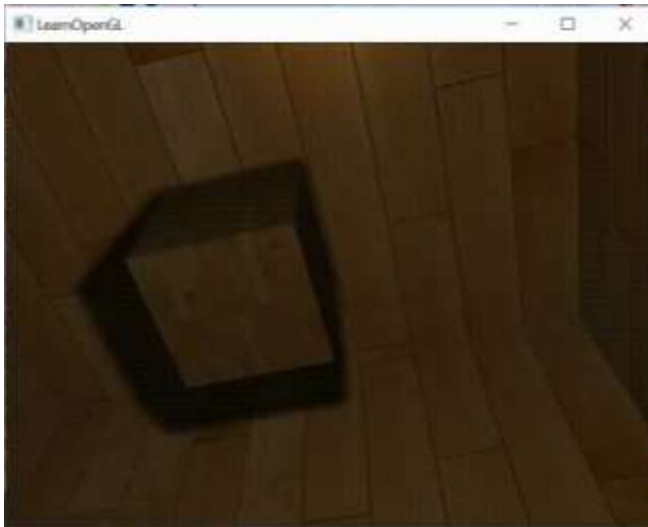
Invece per la matrice di vista, si ha bisogno di una diversa per ogni direzione. Con `glm::lookAt` creiamo 6 direzioni di vista, ognuna delle quali guarda una direzione della faccia della cubemap nell'ordine: destra, sinistra, in alto, in basso, vicino e lontano. Qui creiamo 6 matrici di vista e le moltiplichiamo con la matrice di proiezione per ottenere un totale di 6 diverse matrici di trasformazione dello spazio luminoso.

Queste matrici di trasformazione vengono inviate agli shader che eseguono il rendering della profondità nella cubemap.

Per renderizzare i valori di profondità in una Cubemap di profondità si ha bisogno di un totale di tre shader: uno shader per i vertici e per i frammenti e uno shader per la geometria.

Lo shader geometrico sarà lo shader responsabile della trasformazione di tutti i vertici del world-space ai 6 diversi spazi di luce. Pertanto, lo shader dei vertici si limita a trasformare i vertici nello spazio del mondo e li indirizza allo shader geometrico.

Lo shader geometrico riceve in input 3 vertici di triangolo e una matrice uniforme di matrici di trasformazione dello spazio di luce. Lo shader geometrico è responsabile della trasformazione dei vertici negli spazi di luce. (File [Point_shadow](#))



Una volta eseguita la tecnica, ci si accorge però che le ombre risultano frastagliate. Perciò viene implementata la Percentage-Close Filtering (PCF) che permette di rendere le ombre molto più smooth, poiché si va a calcolare il valore dell'ombra come la media tra la sua ombra e quelle vicine. (File [Point shadow soft](#))