

Introduzione allo Shadow Mapping in Computer Graphics

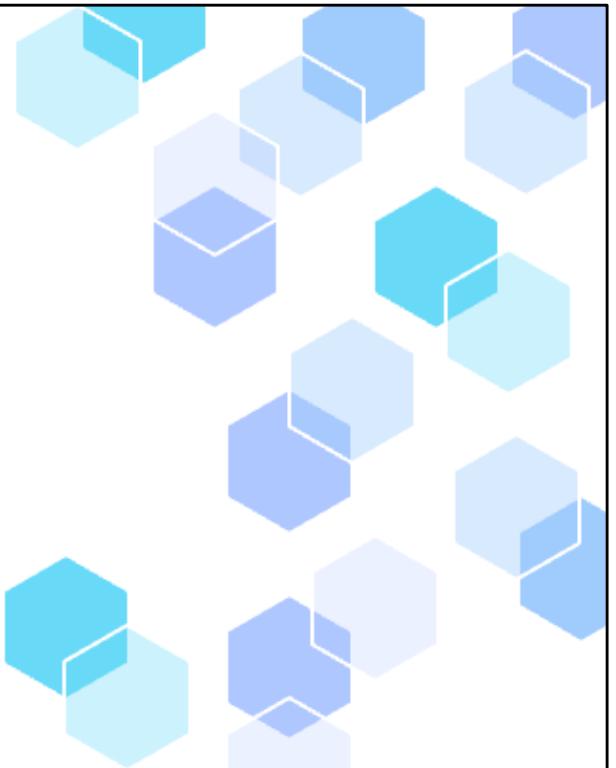
Docente:

Diego Romano

Studente:

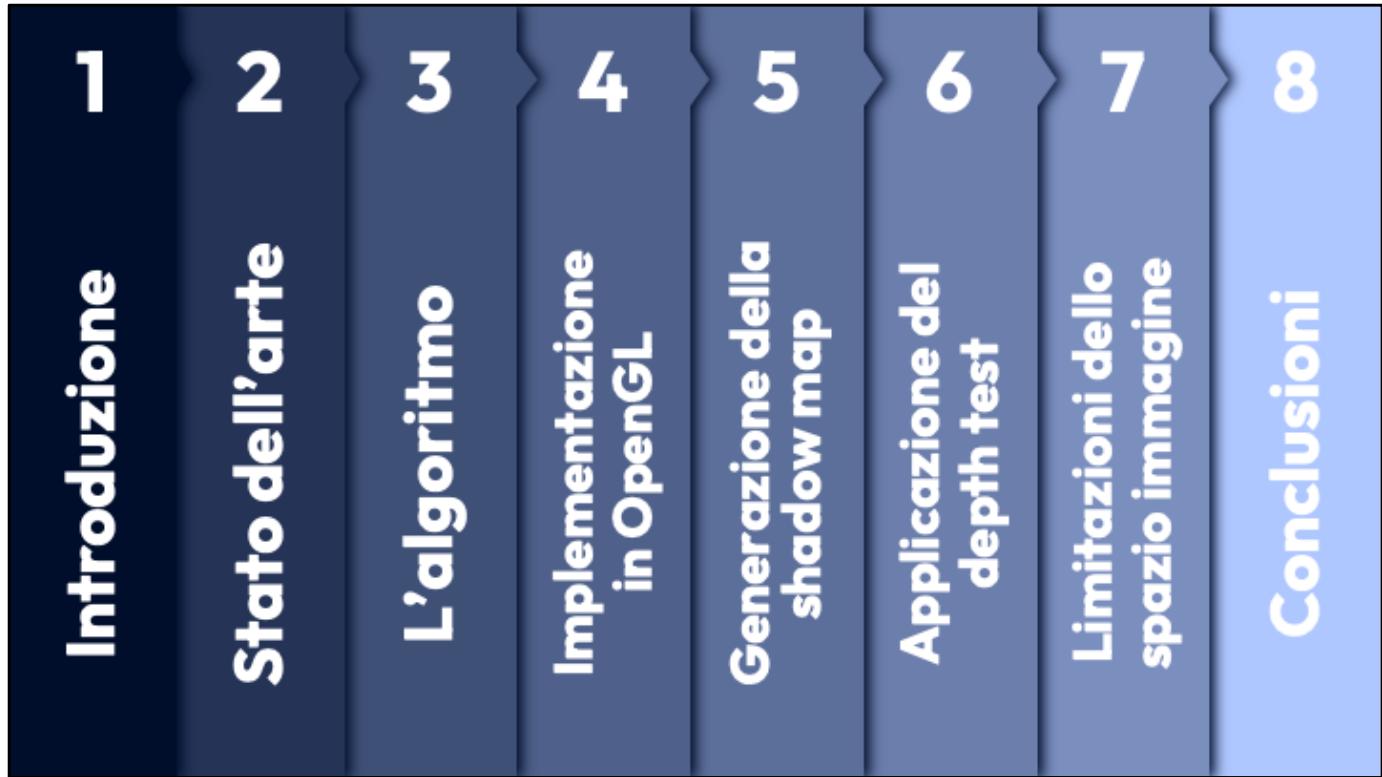
Mario Gabriele Carofano

Università degli Studi di Napoli "Federico II"



Buongiorno!

In questa presentazione cercherò di riassumere il lavoro svolto per l'elaborato finale di Computer Graphics: lo studio della generazione di ombre tramite l'utilizzo della tecnica dello shadow mapping, con uno sguardo attento a vantaggi e svantaggi, al confronto con altre tecniche e, infine, l'implementazione di una demo in OpenGL.

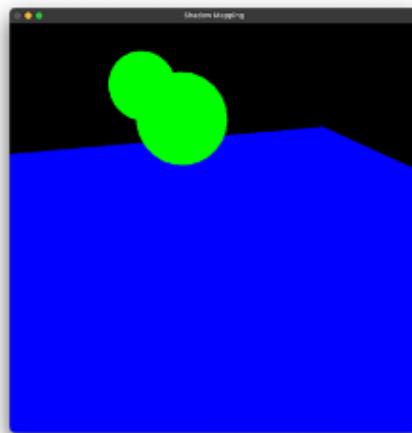


Dividiamo la discussione in queste 8 sezioni:

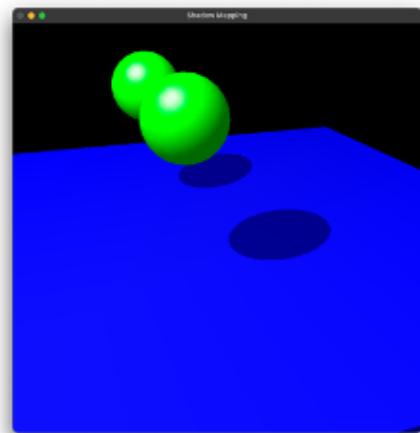
- Si inizia con una breve introduzione al problema nella sezione (1).
- Prima di parlare dell'algoritmo vero e proprio nella sezione (3), andiamo a introdurre nella sezione (2) i concetti di riferimento per una miglior comprensione dell'intera presentazione.
- Quindi, nelle sezioni (4), (5) e (6) trattiamo nel dettaglio l'implementazione della demo in OpenGL e i comandi adoperati per la realizzazione dello shadow mapping.
- Infine, nella sezione (7) una breve carrellata delle criticità di questa tecnica e nella sezione (8) un riassunto sugli esiti e sui possibili spunti di ricerca.

Introduzione

Le ombre nella computer graphics



Rendering di una scena senza illuminazione



Rendering di una scena con illuminazione e shadow mapping

Nell'ambito della computer grafica, le ombre sono utilizzate per migliorare sensibilmente:

- **L'INTELLIGIBILITÀ DI UNA SCENA**, cioè la chiarezza dell'interpretazione.
- **COMPRENSIBILITÀ DI UN OGGETTO** e la sua posizione relativa nella scena rispetto alla fonte luminosa e all'osservatore.
- **IL LIVELLO DI REALISMO** di un'animazione. In particolare, questo punto è di fondamentale importanza in quanto la tecnica dello shadow mapping nasce proprio per migliorare le prestazioni delle animazioni 3D in real-time.

Determinare se un punto sia o meno in ombra non è un'operazione banale:

- a livello tecnico dal punto di vista della potenza di calcolo richiesta
- ma anche a livello implementativo dal punto di vista degli svariati algoritmi che si possono scegliere.

Stato dell'arte

Shadow rendering



Shadow volumes

Frank Crow (1977)

Stencil buffer

Object based

Doom 3 (id Software)



Shadow mapping

Lance Williams (1978)

Depth buffer

Image based

Toy Story (Pixar's Renderman)

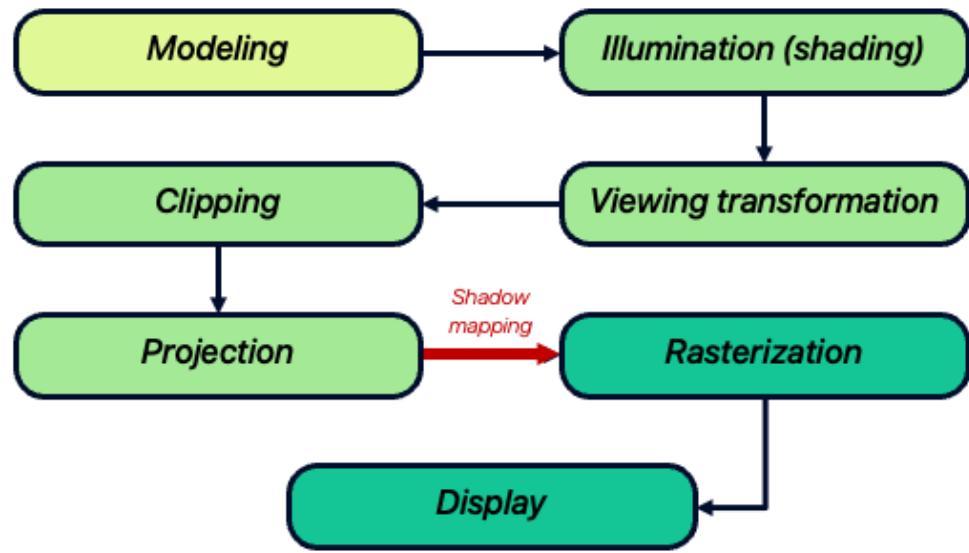
Storicamente, sono stati pubblicati diversi metodi. Tra questi, ricordiamo:

- L'algoritmo per gli SHADOW VOLUMES,
1. introdotto da Frank Crow nel 1977
 2. utilizza una mesh di poligoni che proiettano l'ombra degli oggetti andando ad estrudere la geometria (o silhouette) dei vertici che sono illuminati.
 3. Una possibile implementazione per questa tecnica utilizza lo stencil buffer per memorizzare il poligono che deve essere estruso.
 4. Rientra nella categoria delle tecniche object-based, in quanto si basano sulla geometria degli oggetti presenti nella scena, richiedendo maggiore potenza di calcolo all'aumentare della sua complessità ma, allo stesso tempo, per ottenere una miglior qualità delle ombre generate.
- L'algoritmo per lo SHADOW MAPPING
1. introdotto da Lance Williams nel 1978 nel paper "Casting curved shadows on curved surfaces"

2. La sua implementazione prevede l'utilizzo di una texture ottenuta dal depth buffer, detta depth map o shadow map, per disegnare le ombre proiettate da una singola fonte luminosa.
3. lavora in post-produzione, cioè quando la scena è stata già renderizzata. Cioè, non si lavora sullo spazio tridimensionale degli oggetti in scena ma sullo spazio bidimensionale dello schermo.
4. Per questo motivo, rientra nella categoria delle tecniche image-based.

Stato dell'arte

Rendering pipeline



La pipeline grafica è la sequenza di operazioni che restituisce un'immagine bitmap (a seguito della fase di display) a partire dagli oggetti tridimensionali presenti in una scena (fase di modeling). Tra queste due fasi, evidenziata in verde chiaro, vi è quella di rendering, che trasforma lo spazio tridimensionale in input in un'immagine in uno spazio bidimensionale.

Come detto prima, lo shadow mapping lavora in post-produzione, cioè esattamente prima della fase di «Rasterization». In questo modo, può offrire delle prestazioni efficienti siccome non dipende dalla complessità della scena (numero di oggetti), ma solo dalla risoluzione dell'immagine (da qui il termine image-based). Tuttavia, come vedremo più avanti, questa caratteristica presenta anche alcuni svantaggi, tra cui vorrei citare gli errori di quantizzazione, l'aliasing e l'oscuramento errato di alcuni pixel.

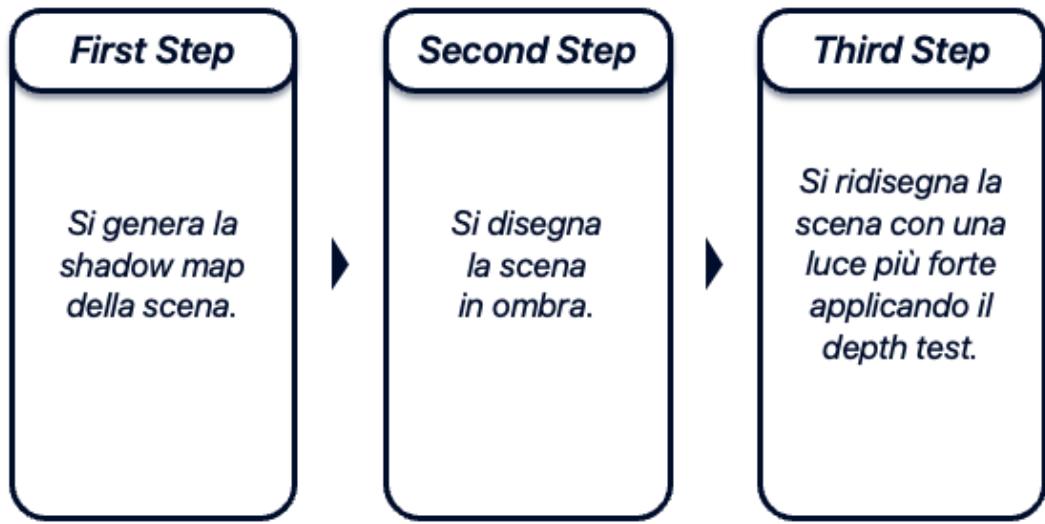


Esempio di depth map

Nella realizzazione dello shadow mapping, quindi, è di fondamentale importanza capire cos'è e come funziona una shadow map.

Una shadow map altro non è che un'immagine contenente la misura della distanza dei punti di un oggetto da un determinato punto di vista. Si utilizza il depth buffer per memorizzare questi valori. Vediamo in questo esempio, una scacchiera le cui pedine sono posizionate a distanze diverse.

Una shadow map si può rappresentare graficamente in scala di grigi, assegnando ai punti della scena più distanti delle gradazioni scure, mentre a quelli più vicini delle gradazioni chiare, proprio come in questo esempio.



Passiamo all'algoritmo vero e proprio introdotto da Lance Williams.

In esso, Williams tenta proprio di sfruttare le informazioni offerte dalla depth map restituita dal calcolo della visibilità tramite depth buffer. Lo schema esecutivo, quindi, può essere riassunto nei seguenti 3 passaggi.

1. *Si disegna la scena dal punto di vista della fonte luminosa per generare la shadow map.*
2. *Si disegna la scena dal punto di vista dell'osservatore con una luce debole per ottenere l'effetto degli oggetti «in ombra».*
3. *Si ridisegna la scena dal punto di vista dell'osservatore attivando l'illuminazione locale e si applica il depth test per capire quali punti sono in ombra e quali devono essere illuminati.*

First Step

Si disegnano solo le facce esterne degli oggetti in scena dal **punto di vista della luce**.

Si memorizza nel **depth buffer** l'informazione di profondità (valore z) solo per le facce interne.

Si converte il **buffer in una texture** delle stesse dimensioni della finestra.

Per quanto riguarda il first step...

Questa scelta algoritmica è data dalla necessità di dover ridurre gli errori di quantizzazione. Con questo termine si intende il peggioramento della qualità delle ombre generate dovuto alla bassa precisione aritmetica del depth buffer. Per alleviare questo tipo di errori, una possibile soluzione, quindi, è quella appena proposta, cioè:

- disegnare le facce esterne
- memorizzare l'informazione di profondità delle sole facce interne.

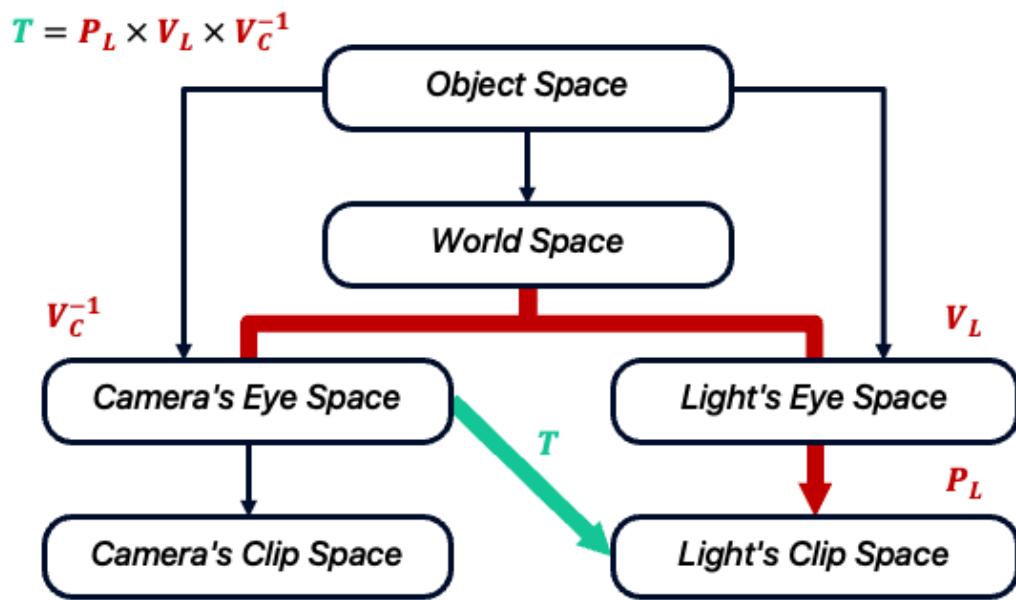
Infine...

Second Step

→ *Si disegna la scena dal punto di vista della camera.*

→ *Si utilizza una fonte luminosa molto debole per ottenere l'effetto degli oggetti in ombra.*

A proposito del second step...



Prima di introdurre il third step, vorrei focalizzare l'attenzione sul funzionamento del depth test. Infatti, per la buona riuscita dell'algoritmo, è necessario proiettare il depth buffer (come texture) ottenuto dalla scena disegnata dal punto di vista della fonte luminosa NELLA scena disegnata dal punto di vista della camera.

Per eseguire la proiezione della shadow map, utilizziamo la seguente trasformazione lineare che converte la matrice delle coordinate della texture calcolate nel Light's Clip Space alle coordinate del Camera Eye's Space, passando solo per il "World Space" e il "Light's Eye Space". Ciò significa che l' Object Space non deve essere ricalcolato ad ogni redraw (infatti, abbiamo già detto in precedenza che lo shadow mapping è una tecnica image-based).

Third Step

- Si disegna la scena dal **punto di vista della camera**, utilizzando una **luce bianca** per ottenere l'effetto degli oggetti illuminati.
- Preso un qualsiasi punto dell'immagine della scena, si confronta il valore D memorizzato nel depth buffer con il valore R della distanza tra il punto scelto e la luce.
- Se $R = D$, allora **il punto non è in ombra**. Si disegna solo la scena illuminata.
- Se $R > D$, allora **il punto è in ombra**. Si sovrappone la scena in ombra a quella illuminata.

Quindi, per il third step ...

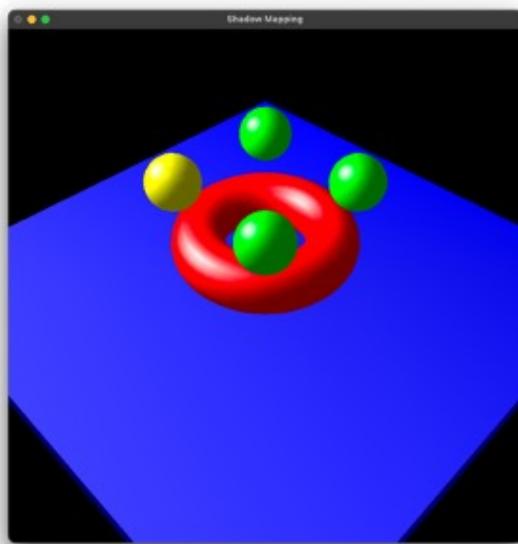
Se $R = D$, allora **il punto non è in ombra**. Non vi è nessun ostacolo nel percorso tra la luce e quest'ultimo.

Se $R > D$, allora **il punto è in ombra**. C'è almeno un oggetto che ostacola l'arrivo della luce sul punto in esame. In particolare, vedremo nell'implementazione che il test significativo è questo e non quello di uguaglianza, siccome è meno probabile che restituisca risultati errati a causa della precisione del depth buffer.

Per tutti i punti per cui vale $R \leq D$, le superfici visibili alla luce saranno sicuramente non in ombra. In particolare, vale per $R = D$ per la giustificazione data prima e vale per $R < D$ perché sono superfici interne nascoste alla luce per definizione (cioè, questo valore di confronto non è significativo).

Implementazione in OpenGL

Codice della Scena 1



```
glColor3fv(BLUE);
glScalef(1.0f, 0.05f, 1.0f);
glutSolidCube(4.0f);

glColor3fv(RED);
glTranslatef(0.0f, 0.5f, 0.0f);
glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
glutSolidTorus(0.2, 0.5, 24, 48);

glRotatef(angle, 0.0f, 1.0f, 0.0f);
glColor3fv(GREEN);

glTranslatef(0.45f, 1.0f, 0.4f);
glutSolidSphere(0.2, 24, 24);

glTranslatef(-0.9f, 0.0f, 0.0f);
glutSolidSphere(0.2, 24, 24);

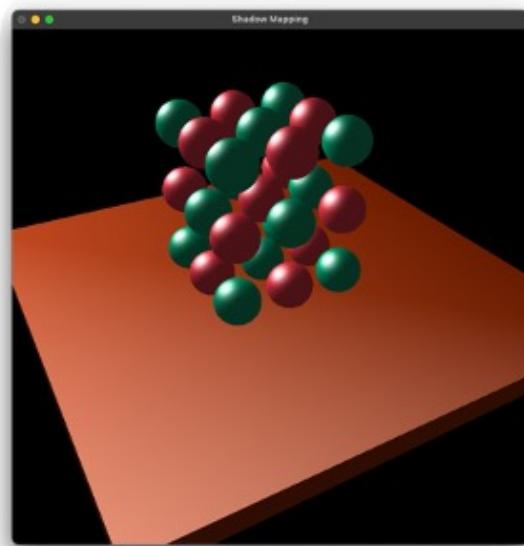
glTranslatef(0.0f, 0.0f,-0.9f);
glutSolidSphere(0.2, 24, 24);

glColor3fv(YELLOW);
glTranslatef(0.9f, 0.0f, 0.0f);
glutSolidSphere(0.2, 24, 24);
```

Il programma che ho realizzato per implementare l'algoritmo dello shadow mapping utilizza le librerie GLFW3 e GLUT per l'esecuzione di comandi OpenGL tramite CPU. Le scene che andremo ad esaminare nelle prossime slides sono le seguenti.

Implementazione in OpenGL

Codice della Scena 2



```
glColor3fv(BROWN);
glScalef(1.0f, 0.05f, 1.0f);
glutSolidCube(4.5f);

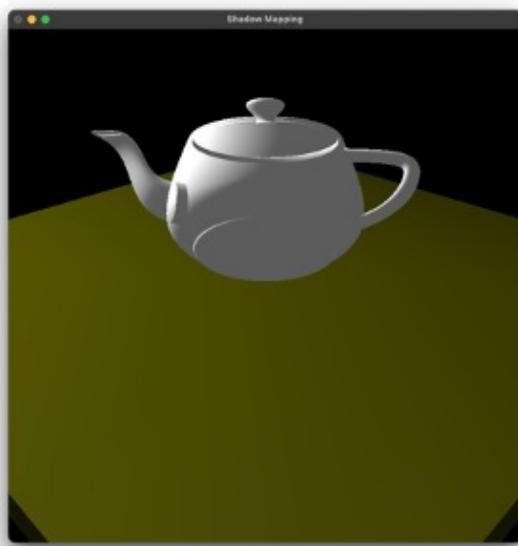
glTranslatef(-0.7f, 0.5f, -0.7f);

for (l1 = 0; l1 < 3; l1++) {
    glPushMatrix();
    for (l2 = 0; l2 < 9; l2++) {
        if ((l1+l2)%2 == 0)
            glColor3fv(CHRISTMAS_GREEN);
        else
            glColor3fv(CHRISTMAS_RED);
        glutSolidSphere(0.2, 24, 24);
        if (l2%3 != 2)
            glTranslatef(0.0f, 0.0f, 0.5f);
        else
            glTranslatef(0.5f, 0.0f, -1.0f);
    }
    glPopMatrix();
    glTranslatef(0.0f, 0.6f, 0.0f);
}
```

Una famosa scena proposta dal produttore di schede grafiche NVIDIA.
<https://web.archive.org/web/20040619120702/http://developer.nvidia.com/attach/6769>

Implementazione in OpenGL

Codice della Scena 3



```
glColor3fv(YELLOW_20);
glScalef(1.0f, 0.05f, 1.0f);
glutSolidCube(4.0f);
```

```
glTranslatef(0.0f, 0.7f, 0.0f);
glRotatef(40.0f, 0.0f, 1.0f, 0.0f);
glColor3fv(GRAY_90);
glutSolidTeapot(0.6);
```

Una scena con il modello tridimensionale della teiera, famosissimo nell'ambito della computer grafica.

Le immagini mostrate sono state renderizzate volontariamente solo con l'illuminazione locale e senza generazione delle ombre. Infatti, proprio per questo motivo, è quasi impossibile riuscire a capire se la teiera è appoggiata o meno sulla base, ad esempio.

```
glClearDepth(1.0f);
glDepthFunc(GL_LESS);
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_CULL_FACE);
```

```
 glGenTextures(1, &shadowMapTexture);
 glBindTexture(GL_TEXTURE_2D, shadowMapTexture);
 glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, shadowMapSize,
 shadowMapSize, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

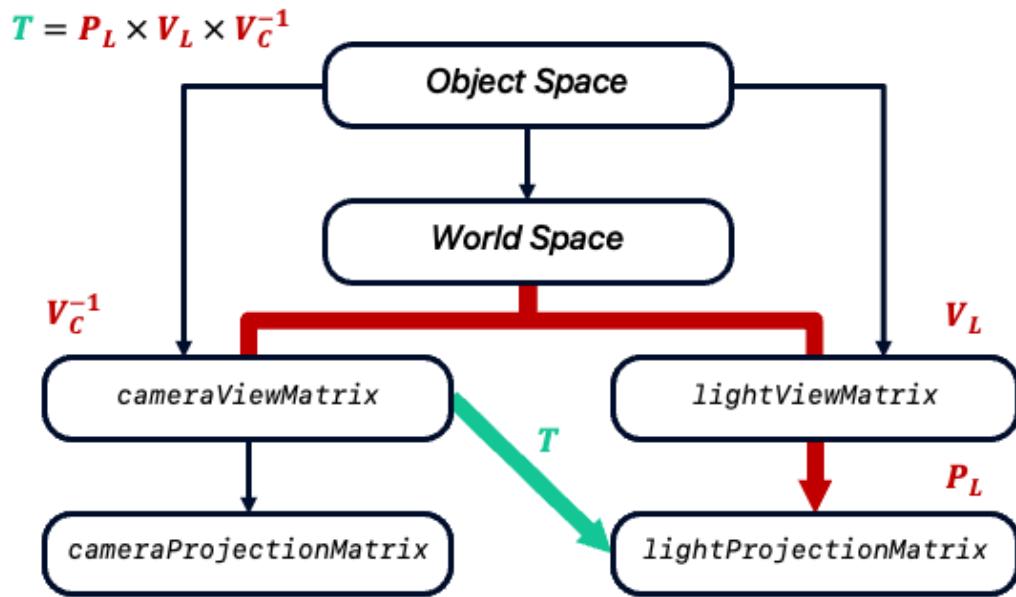
Per far funzionare correttamente il tutto, è stato necessario inizializzare l'ambiente di lavoro utilizzando i seguenti comandi:

- `glEnable(GL_DEPTH_TEST)` . Per abilitare il depth test, cioè il confronto tra la distanza di un pixel dalla camera e quella memorizzata nel depth buffer.
- `glDepthFunc(GL_LESS)` . Specifica quale funzione utilizzare per il confronto della profondità; in questo caso, il test termina con successo se la distanza $R \leq D$ (e quindi il pixel si illumina).
- `glEnable(GL_CULL_FACE)` . Per abilitare il comando `glCullFace()` durante la generazione della shadow map, utile per disegnare nella texture solo le facce posteriori.
- `glGenTextures()` . Per generare un identificativo numerico alla texture della shadow map.
- `glBindTexture()` . Per associare la `GL_TEXTURE_2D` all'identificativo `shadowMapTexture`.
- `glTexImage2D()` . Per specificare la texture bidimensionale. La dimensione è data dalla variabile `shadowMapSize`, mentre `GL_DEPTH_COMPONENT` indica che l'informazione memorizzata nella texture non è un colore ma un'informazione di profondità nel range $[0,1]$.
- `glTexParameteri()` . Per specificare alcuni parametri di tipo intero per la texture

(es. per specificare il comportamento del programma in caso di magnification / minification, e il comportamento della texture quando non rientra nel range [0,1] - in questo caso GL_CLAMP_TO_EDGE significa che si ripete solo l'ultimo pixel).

Implementazione in OpenGL

Inizializzazione per il depth test



Riprendiamo velocemente lo schema per l'implementazione della trasformazione lineare. Come visto prima, la realizzazione delle ombre tramite la tecnica dello shadow mapping ha bisogno di quattro matrici.

1. *cameraProjectionMatrix*
2. *cameraViewMatrix*
3. *lightProjectionMatrix*
4. *lightViewMatrix*

Le matrici con denominazione "projection" si riferiscono alle matrici di proiezione, utilizzate per inizializzare le trasformazioni di visualizzazione (es. proiezione prospettica, ortografica, ...). Invece, le matrici con denominazione "view" si riferiscono alle matrici di modellazione, utilizzate per inizializzare le trasformazioni di modellazione (es. redraw della scena con rotate, scale, translate, ...).

Ognuna di queste è inizializzata tramite i seguenti comandi.

```
glLoadIdentity();
gluPerspective(45.0f, (float)
windowWidth/windowHeight, 1.0f, 100.0f);
glGetFloatv(GL_PROJECTION_MATRIX,
cameraProjectionMatrix);
```

Aggiornamento di cameraProjectionMatrix

- `gluPerspective()` per specificare e ottenere una proiezione prospettica in un volume di vista in cui il piano di proiezione è dato dal rapporto delle dimensioni del viewport, mentre i valori di `nearClip` e `farClip` sono rispettivamente `1.0f` e `100.0f`
- `glGetFloatv()` per aggiornare i valori della matrice selezionata.

Implementazione in OpenGL

Inizializzazione per il depth test

```
GLfloat cameraPositions[3][3] = {  
    {-2.5f, 3.5f, -2.5f}, // SCENE1  
    {-4.5f, 3.5f, -2.5f}, // SCENE2  
    {-2.5f, 2.0f, -2.5f} // SCENE3  
};
```

```
glLoadIdentity();  
gluLookAt(  
    cameraPositions[scene-1][0], cameraPositions[scene-1][1],  
    cameraPositions[scene-1][2],  
    0.0f, 0.0f, 0.0f,  
    0.0f, 1.0f, 0.0f);  
glGetFloatv(GL_MODELVIEW_MATRIX, cameraViewMatrix);
```

Aggiornamento di cameraViewMatrix

- `gluLookAt()` per costruire una nuova matrice di visualizzazione a partire da un punto di vista. In questo caso, le coordinate dell'eyepoint sono diverse per ogni scena e sono memorizzate nella matrice 3×3 `cameraPositions`.

```
glLoadIdentity();
gluPerspective(45.0f, 1.0f, 2.0f, 8.0f);
glGetFloatv(GL_PROJECTION_MATRIX, lightProjectionMatrix);
```

Aggiornamento di lightProjectionMatrix

Note.

Implementazione in OpenGL

Inizializzazione per il depth test

```
VECTOR3D lightPosition(2.0f, 3.0f, -2.0f);
```

```
glLoadIdentity();
gluLookAt(
    lightPosition.GetX(),
    lightPosition.GetY(),
    lightPosition.GetZ(),
    0.0f, 0.0f, 0.0f,
    0.0f, 1.0f, 0.0f
);
glGetFloatv(GL_MODELVIEW_MATRIX, lightViewMatrix);
```

Aggiornamento di lightViewMatrix

Utilizza la stessa posizione della fonte luminosa per tutte le 3 scene.

5

Generazione della shadow map

FirstStep() - Codice

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
eyePosition = EYE_LIGHT;
glMatrixMode(GL_PROJECTION);
glLoadMatrixf(lightProjectionMatrix);
glMatrixMode(GL_MODELVIEW);
glLoadMatrixf(lightViewMatrix);
glViewport(0, 0, shadowMapSize, shadowMapSize);

Codice della funzione UpdateEyePosition()

glCullFace(GL_FRONT);
glShadeModel(GL_FLAT);
DrawScene(angle, scene);

glBindTexture(GL_TEXTURE_2D, shadowMapTexture);
glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, 0, 0,
shadowMapSize, shadowMapSize, 0);
```

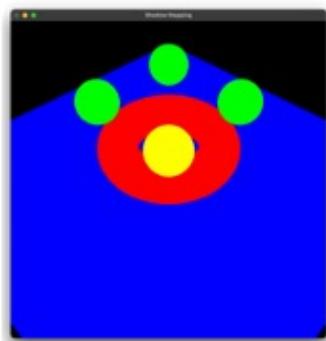
Per il primo passaggio, si inizia con la generazione della shadow map.

- Resetiamo il color buffer ed il depth buffer.
- Aggiorniamo le matrici di proiezione e visualizzazione con la `lightProjectionMatrix` e `lightViewMatrix`, rispettivamente.
- Ridimensioniamo il viewport in modo tale che abbia la stessa dimensione della shadow map.
- Con il comando `glCullFace()` scegliamo di disegnare solo le facce anteriori, in modo che solo quelle posteriori siano memorizzate nella shadow map.
- Dopo aver disegnato la scena con la procedura `DrawScene()`, associamo la texture `GL_TEXTURE_2D` al nostro identificativo e copiamo i valori del depth buffer nella texture selezionata con il comando `glCopyTexImage2D()`, utilizzando sempre come "internalFormat" la costante `GL_DEPTH_COMPONENT`.

5

Generazione della shadow map

FirstStep() - Risultati



Scena 1



Scena 2



Scena 3

Note.

```
glClear(GL_DEPTH_BUFFER_BIT);  
UpdateEyePosition(EYE_CAMERA);  
glLightfv(GL_LIGHT1, GL_POSITION, VECTOR4D(lightPosition));  
glLightfv(GL_LIGHT1, GL_AMBIENT, GRAY_20);  
glLightfv(GL_LIGHT1, GL_DIFFUSE, GRAY_20);  
glLightfv(GL_LIGHT1, GL_SPECULAR, BLACK);  
glEnable(GL_LIGHT1);  
glEnable(GL_LIGHTING);
```

Codice della funzione SecondStep()

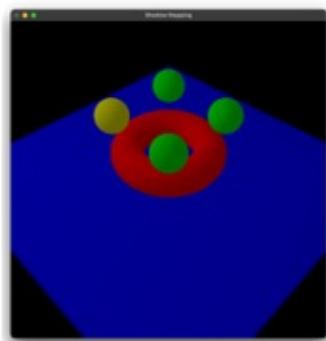
Per il secondo passaggio, invece, disegniamo la scena dal punto di vista della camera utilizzando una luce che dia l'effetto finale delle zone ombreggiate.

- Per iniziare, resettiamo solo il depth buffer, visto che quello del colore non è stato ancora modificato.
- Aggiorniamo le matrici di proiezione e visualizzazione utilizzando quelle della camera.
- Ridimensioniamo il viewport in modo tale che abbia la stessa dimensione della finestra.
- Specifichiamo i parametri per la luce (colore, specularità, posizione) e disegniamo la scena.

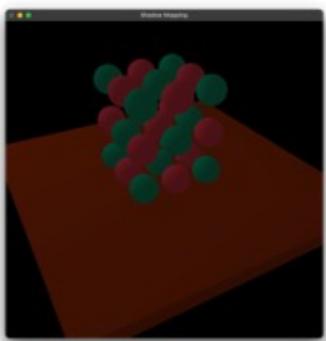
5

Generazione della shadow map

SecondStep() - Risultati



Scena 1



Scena 2



Scena 3

Note.

```

UpdateEyePosition(EYE_CAMERA);
glLightfv(GL_LIGHT1, GL_DIFFUSE, WHITE);
glLightfv(GL_LIGHT1, GL_SPECULAR, WHITE);

MATRIX4X4 textureMatrix = biasMatrix
* lightProjectionMatrix
* lightViewMatrix;

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGenfv(GL_S, GL_EYE_PLANE, textureMatrix.GetRow(0));
glEnable(GL_TEXTURE_GEN_S);

glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGenfv(GL_T, GL_EYE_PLANE, textureMatrix.GetRow(1));
glEnable(GL_TEXTURE_GEN_T);

glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGenfv(GL_R, GL_EYE_PLANE, textureMatrix.GetRow(2));
glEnable(GL_TEXTURE_GEN_R);

glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGenfv(GL_Q, GL_EYE_PLANE, textureMatrix.GetRow(3));
glEnable(GL_TEXTURE_GEN_Q);
glBindTexture(GL_TEXTURE_2D, shadowMapTexture);
glEnable(GL_TEXTURE_2D);

```

Il terzo passaggio è quello dove si implementa il depth test.

Preso un qualsiasi punto degli oggetti visibili nel viewport, se esso supera il depth test, allora deve essere illuminato. Cioè, tecnicamente parlando, deve sovrascrivere il pixel ombreggiato disegnato al passaggio precedente. Se, invece, ci troviamo nel caso contrario, allora si lascia la zona ombreggiata come visto nei risultati precedenti. Quindi:

- Si abilita una luce con riflessione diffusa e speculare di colore bianco.
- Implementiamo la trasformazione lineare, moltiplicando queste tre matrici. In particolare, manca la post-moltiplicazione con la matrice `cameraViewMatrix`: questo perché è un'operazione eseguita in automatico da OpenGL siccome abbiamo aggiornato la matrice `GL_MODELVIEW` con la procedura `UpdateEyePosition()`.
- Da questo passaggio otteniamo la matrice `textureMatrix`, che ha valori nell'intervallo $[-1,1]$. Per modificare il range di valori di questa matrice e ottenere definitivamente la matrice che proietta i texel della shadow map sui pixel della scena, utilizziamo i seguenti tre comandi (una volta per ogni coordinata della texture).

```
glTexParameteri(  
    GL_TEXTURE_2D,  
    GL_TEXTURE_COMPARE_MODE,  
    GL_COMPARE_R_TO_TEXTURE );
```

```
glTexParameteri(  
    GL_TEXTURE_2D,  
    GL_TEXTURE_COMPARE_FUNC,  
    GL_LEQUAL );
```

```
glTexParameteri(  
    GL_TEXTURE_2D,  
    GL_DEPTH_TEXTURE_MODE,  
    GL_INTENSITY );
```

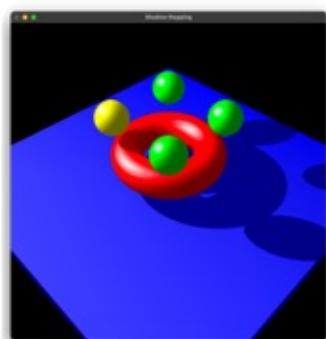
```
glAlphaFunc(GL_GEQUAL, 0.99f);  
glEnable(GL_ALPHA_TEST);  
DrawScene(angle, scene);
```

A questo punto, si specifica la modalità e la funzione di confronto della shadow map utilizzando il comando `glTexParameteri()`.

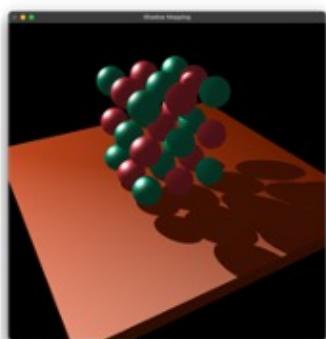
- Il parametro `GL_COMPARE_R_TO_TEXTURE` indica il confronto dei valori R nella texture con i valori D nel depth buffer.
- Il parametro `GL_LEQUAL` indica che la funzione di confronto utilizzata è il minore-uguale, come spiegato nell'algoritmo.
- Il parametro `GL_INTENSITY` indica di trattare il valore 0 (false) o 1 (true) restituito dal depth test come, rispettivamente, un colore nero o bianco sui 4 canali RGBA. Cioè si tiene in considerazione la scena illuminata se si ha "true", oppure si scarta e si ombreggia la scena se si ha "false".
- Infine, per disegnare finalmente la scena illuminata e capire in quali punti lasciare o meno l'ombra, si utilizza un alpha test con limite superiore impostato a 0.99. Se il depth test fallisce (cioè non è 1), non si deve visualizzare il punto illuminato, ma lasciare il punto già renderizzato (in ombra) al passaggio precedente.

Applicazione del depth test

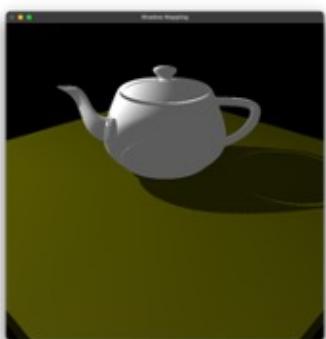
ThirdStep() - Risultato finale



Scena 1



Scena 2



Scena 3

In conclusione, otteniamo i seguenti risultati.

In particolare, si noti come le ombre ottenute da questa tecnica si proiettano sia sui poligoni planari (es. la base) sia sulle superfici curve.

Limitazioni dello spazio immagine

Field of view

Multiple light sources

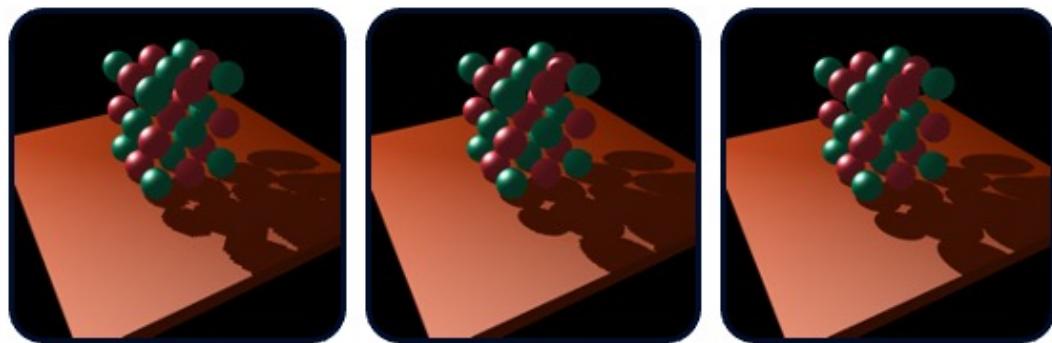
Aliasing

Self shadowing

La generazione di ombre tramite questo approccio porta con sé alcune limitazioni, da analizzare con attenzione e mettere a confronto con i vantaggi.

- **VINCOLO - *Field of view*.** L'algoritmo si basa due viste: la vista dell'osservatore e la vista della fonte luminosa. Un vincolo importante da seguire al fine di ottenere risultati accettabili è quello di garantire che tutti gli oggetti che devono generare un'ombra nel volume di vista dell'osservatore siano all'interno del volume di vista della fonte luminosa (perché la shadow map è calcolata proprio a partire da essa).
- **SVANTAGGIO - *Multiple light sources*.** Nel caso in cui siano presenti un maggior numero di luci in scena, è necessario eseguire l'algoritmo per lo shadow mapping per ogni fonte luminosa. Si ha lo stesso svantaggio anche nel caso in cui sia presente una fonte luminosa omnidirezionale (puntiforme o spotlight) da sezionare su più viste. Ovviamente, in casi del genere questa tecnica non è più

così efficiente.



Risoluzione 128x128

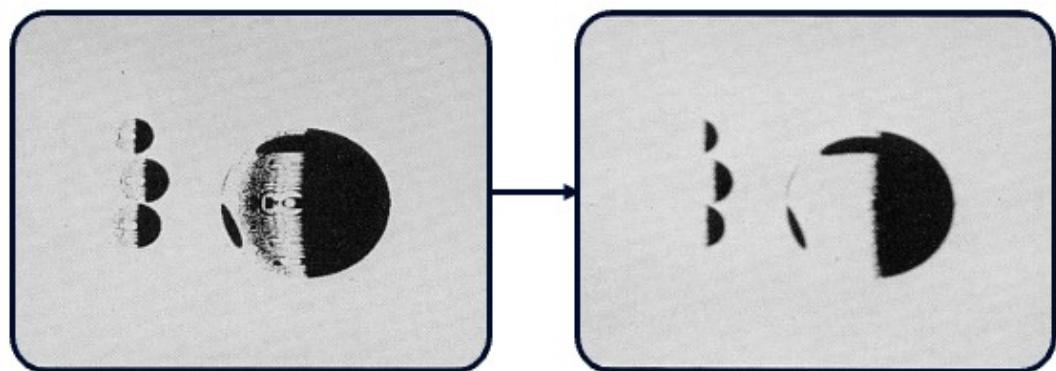
Risoluzione 256x256

Risoluzione 512x512

Con il termine aliasing si intende quel problema di rendering per cui la trasformazione lineare utilizzata nell'applicazione della shadow map accresce in modo errato la dimensione dei suoi texel in prossimità della camera dell'osservatore, ottenendo, quindi, un effetto seghettato.

Esistono diverse tecniche di anti-aliasing che si possono applicare: la più semplice di queste è quella di utilizzare texture più grandi che, ovviamente, richiedono maggior potenza di calcolo.

Queste immagini mostrano come la risoluzione della shadow map è un fattore determinante per la risoluzione di questo problema e, in generale, per ottenere delle ombre di qualità maggiore. Tuttavia, la risoluzione della shadow map non può superare le dimensioni del viewport, altrimenti la trasformazione lineare non restituisce i risultati desiderati.



Applicazione del "biasing" della shadow line

Nell'andare a confrontare il valore di profondità memorizzato nel depth buffer con il valore nella shadow map, vi è una buona probabilità di riscontrare il problema del self shadowing, un problema nel quale i poligoni degli oggetti in scena tendono, appunto, ad ombreggiarsi l'uno con l'altro ottenendo questo effetto di interferenza (moire).

Una soluzione proposta da Williams è quella di muovere la "shadow line", cioè spostare i pixel della scena verso la fonte luminosa solo durante lo shadow test (operazione di biasing). Ecco, quindi, il perché abbiamo utilizzato "biasMatrix" nell'implementazione della trasformazione lineare.

Conclusioni

Utilizzi

- Introdotta da **Lance Williams** nel 1978, nel paper **"Casting curved shadows on curved surfaces"**.
- Viene ancora oggi largamente utilizzata per il **real-time rendering** (es. motore grafico *Renderman* della Pixar).
- Opera con successo su scene in cui sono presenti delle **superfici curve**.

Per ricapitolare il tutto, quindi ...

Tempi di esecuzione

Tempo di inizializzazione		
$1.16 \times 10^{-1} \text{ sec}$		
Tempo di rendering della Scena 1		
Senza ombre	Con shadow mapping	Confronto
$1.49 \times 10^{-2} \text{ sec}$	$1.53 \times 10^{-2} \text{ sec}$	2.69%
Tempo di rendering della Scena 2		
Senza ombre	Con shadow mapping	Confronto
$1.39 \times 10^{-2} \text{ sec}$	$1.42 \times 10^{-2} \text{ sec}$	1.99%
Tempo di rendering della Scena 3		
Senza ombre	Con shadow mapping	Confronto
$1.26 \times 10^{-2} \text{ sec}$	$1.38 \times 10^{-2} \text{ sec}$	8.37%

Per l'implementazione della demo che ho appena presentato vorrei mostrare anche alcuni dati riguardo i tempi di esecuzione.

- Il tempo di inizializzazione riguarda l'esecuzione delle operazioni di inizializzazione che non riguardano propriamente l'applicazione dell'algoritmo per lo shadow mapping, ma solo la fase preliminare.
- Per quanto riguarda i tempi di rendering delle scene, invece, ho misurato il tempo di esecuzione per la scena senza shadow mapping (solo con l'illuminazione locale) e con shadow mapping, ottenendo per la scena 1 una perdita del 2.69%, per la scena 2 una perdita del 1.99% e, infine, per la scena 3 una perdita del 8.37%.

Vantaggi

- Essendo una **tecnica image-based**, il tempo di esecuzione non si basa sulla **complessità della scena**.
- Utilizza una texture, detta "**shadow map**", per memorizzare **velocemente** e in modo **efficace** le informazioni di **shading** di una fonte luminosa.
- Non richiede l'utilizzo dello **stencil buffer** per il calcolo degli "**shadow volumes**", che può **appesantire le prestazioni finali**.
- La **trasformazione lineare** utilizzata per la **proiezione della shadow map** ha un costo che dipende dalla **risoluzione del viewport**.

Abbiamo visto che la tecnica dello shadow mapping, ...

Svantaggi

- Si possono presentare **problemi di aliasing** nell'utilizzare **texture a bassa risoluzione** per la **shadow map**.
- È necessario **renderizzare la scena per ogni fonte luminosa** e ottenere una **shadow map** diversa per ogni luce (**o in caso di luci puntiformi**).
- La fonte luminosa può **generare le ombre** solo all'interno del **volume di vista della fonte luminosa**. È fondamentale **garantire** che tutti gli oggetti da ombreggiare rispettino questo **vincolo**.
- L'applicazione della trasformazione lineare, essendo **in post-produzione**, può scurire **erroneamente** l'illuminazione dei **punti nascosti alla fonte luminosa**.

Dal punto di vista degli svantaggi, invece, abbiamo visto che...

Futuri sviluppi

- *Implementare il depth test utilizzando comandi OpenGL per la GPU, ad esempio gestendo il **fragment shader** ed il **vertex shader**.*
- *Renderizzare la **shadow map** generata dal punto di vista della luce memorizzata nel depth buffer.*
- *Migliorare la **qualità** delle ombre generate utilizzando tecniche di **anti-aliasing**, come il **PCM (Percentage Closer Filtering)** o le **PSM (Perspective Shadow Maps)**.*

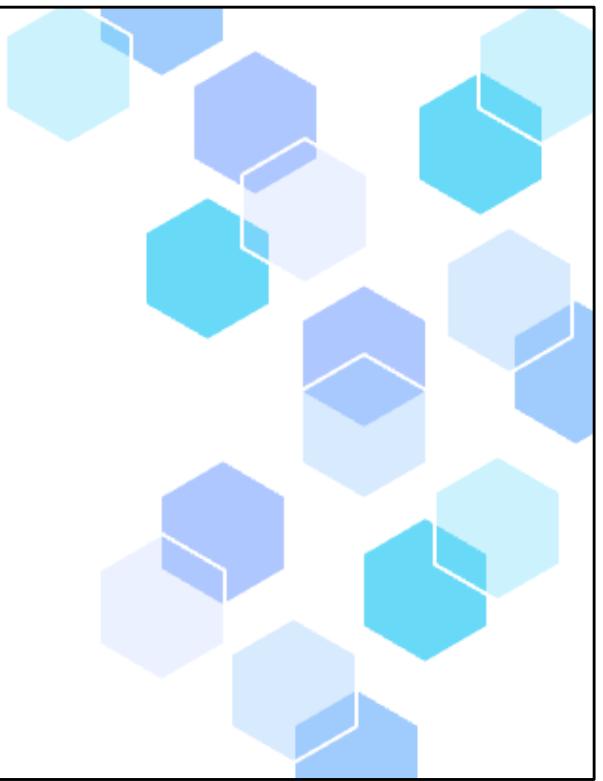
Per continuare la ricerca e l'approfondimento di questa tecnica, sicuramente si può ...

Grazie!

Repository:



<https://github.com/mgcarofano/Shadow-Mapping>



Grazie, e 30 e lode!

Presentazione offerta da Mario Gabriele Carofano.