
Introduzione allo Shadow Mapping in Computer Graphics

Docente:

Diego Romano

Studente:

Mario Gabriele Carofano

Università degli Studi di Napoli "Federico II"



Introduzione

1

Stato dell'arte

2

L'algoritmo

3

Implementazione in OpenGL

4

Generazione della shadow map

5

Applicazione del depth test

6

Limitazioni dello spazio immagine

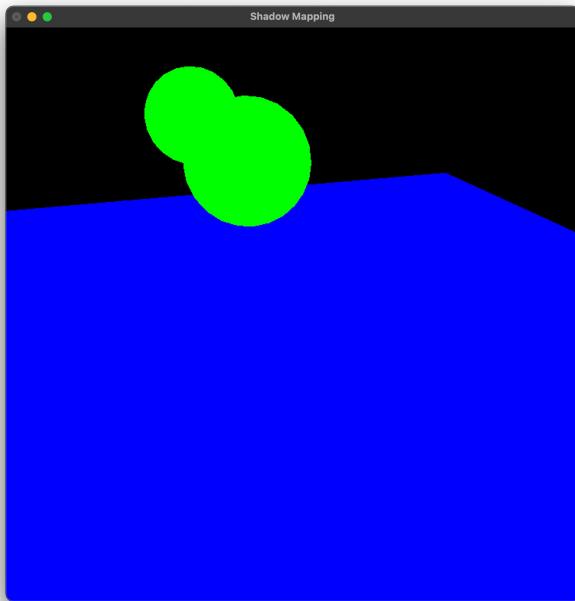
7

Conclusioni

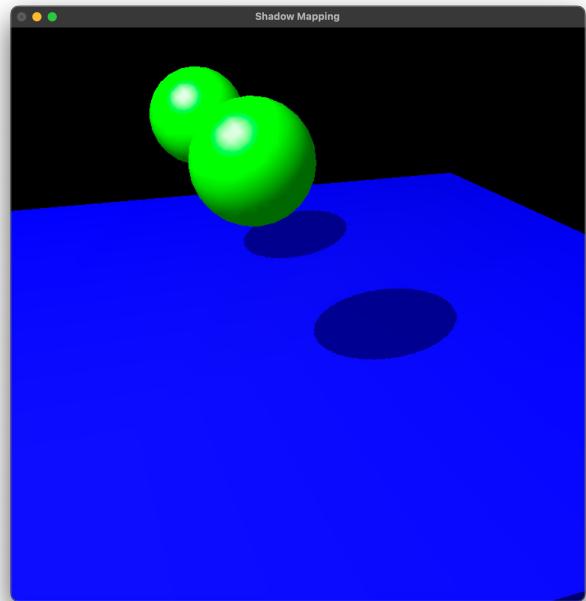
8

Introduzione

Le ombre nella computer graphics



Rendering di una scena senza illuminazione



Rendering di una scena con illuminazione e shadow mapping

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



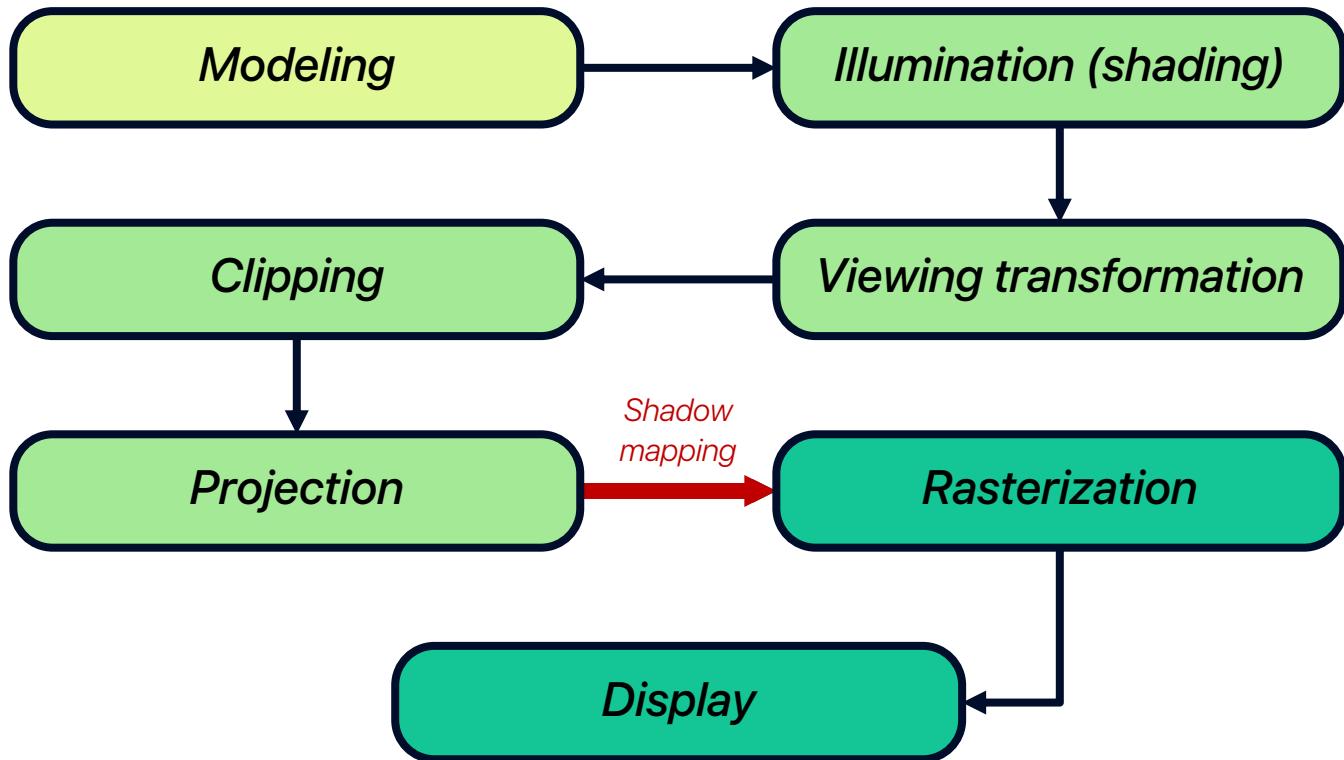
Image based



Toy Story (Pixar's Renderman)

Stato dell'arte

Rendering pipeline



Stato dell'arte

Le depth map



Esempio di depth map

L'algoritmo

Schema esecutivo

First Step

Si genera la shadow map della scena.

Second Step

Si disegna la scena in ombra.

Third Step

Si ridisegna la scena con una luce più forte applicando il depth test.

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.

Second Step

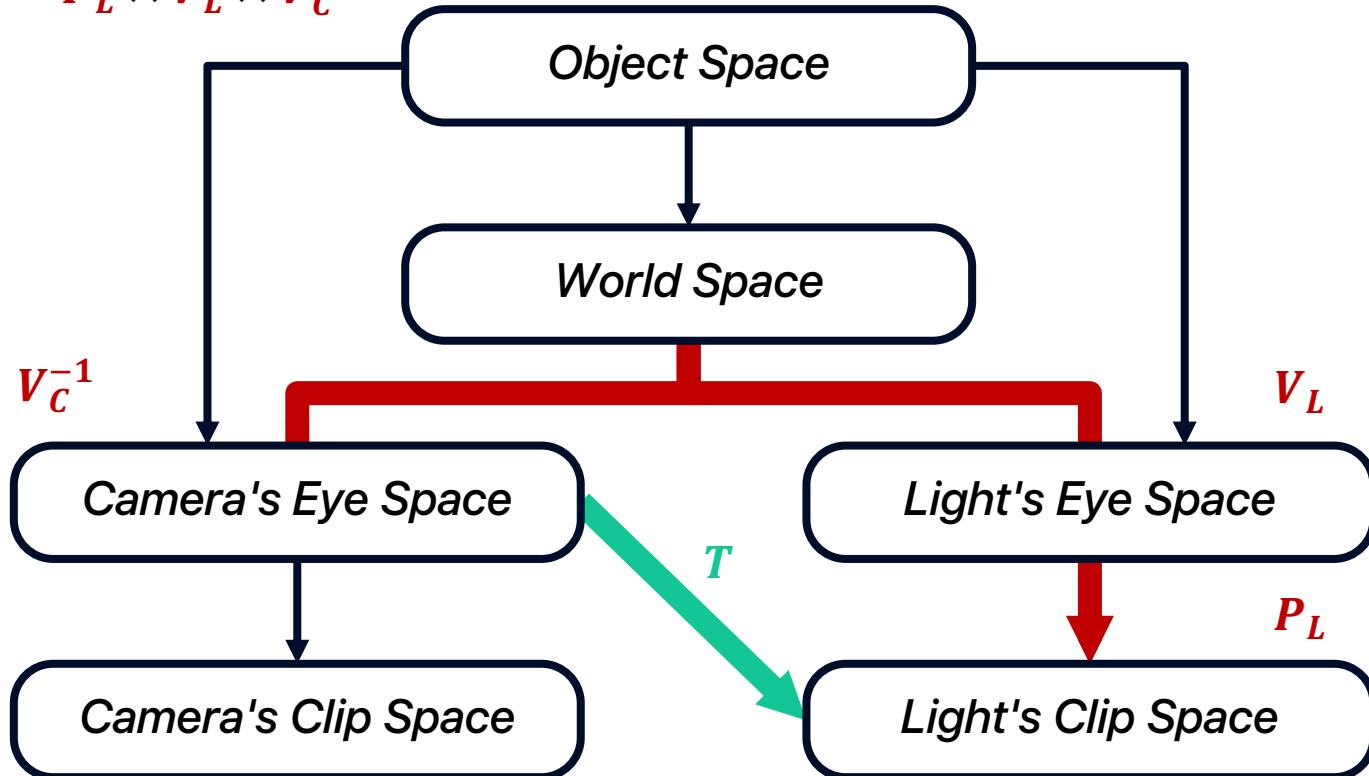
→ Si disegna la scena dal **punto di vista della camera**.

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

L'algoritmo

Schema esecutivo

$$T = P_L \times V_L \times V_C^{-1}$$

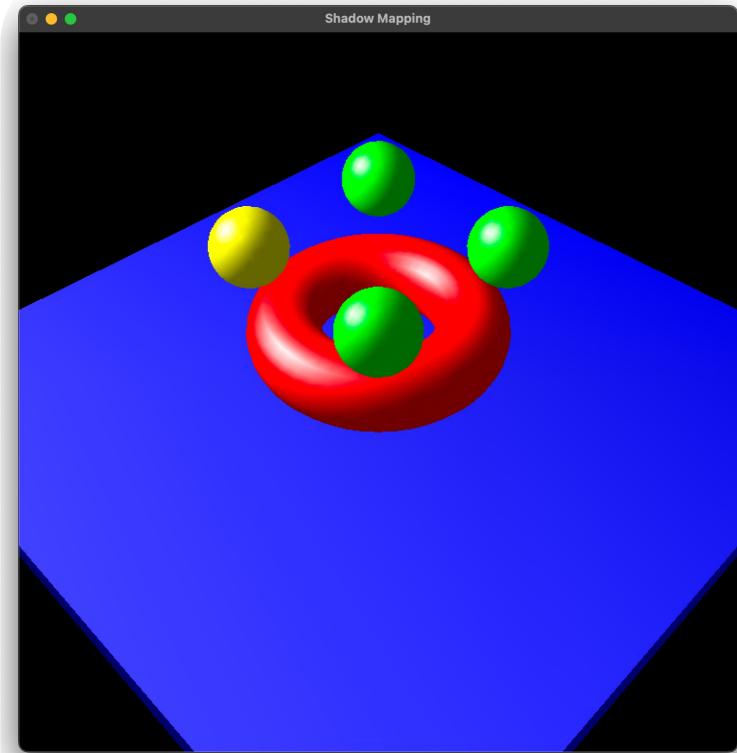


Third Step

- 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.

Implementazione in OpenGL

Codice della Scena 1



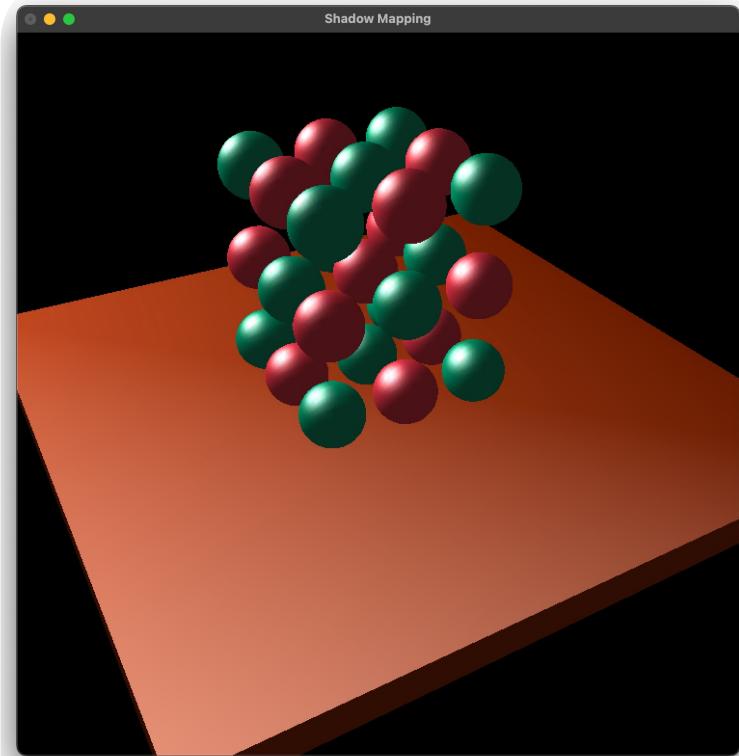
```
glColor3fv(BLUE);  
glScalef(1.0f, 0.05f, 1.0f);  
glutSolidCube(4.0f);
```

```
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, 0.4);  
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);
```

Implementazione in OpenGL

Codice della Scena 2



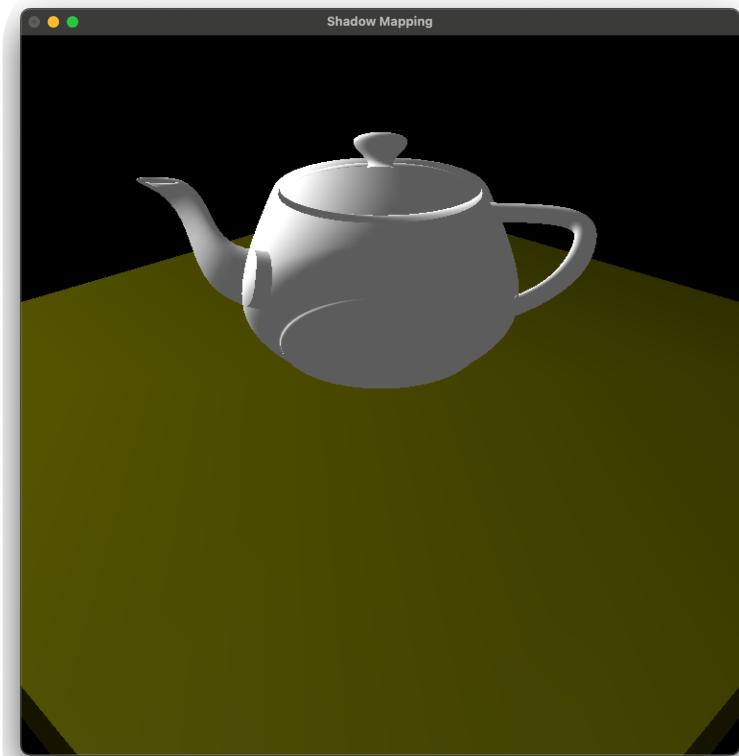
```
glColor3fv(BROWN);
glScalef(1.0f, 0.05f, 1.0f);
glutSolidCube(4.5f);

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);
}
```

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);
```

Implementazione in OpenGL

Inizializzazione per il depth test

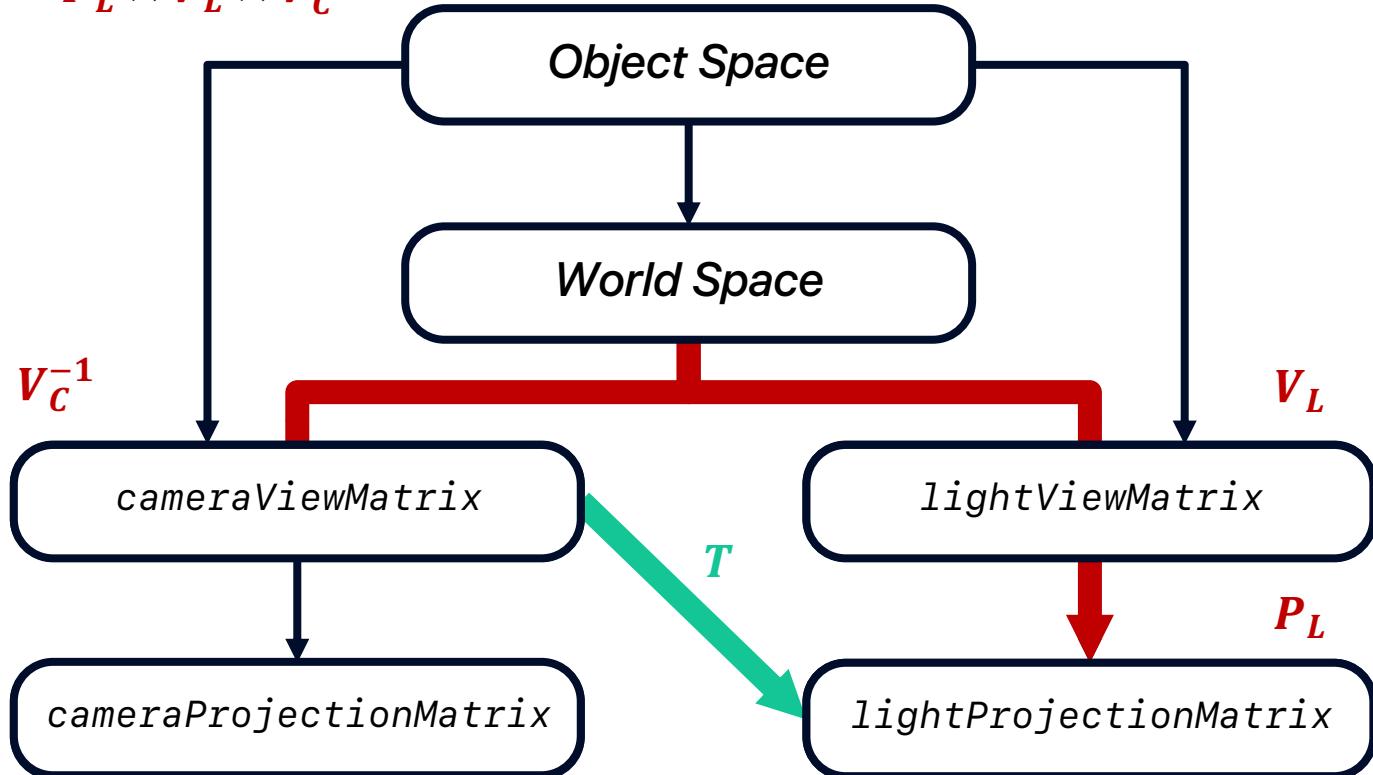
```
glClearDepth(1.0f);  
glDepthFunc(GL_LEQUAL);  
 glEnable(GL_DEPTH_TEST);  
 glEnable(GL_CULL_FACE);
```

```
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);
```

Implementazione in OpenGL

Inizializzazione per il depth test

$$T = P_L \times V_L \times V_C^{-1}$$



Implementazione in OpenGL

Inizializzazione per il depth test

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

Aggiornamento di cameraProjectionMatrix

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*

Implementazione in OpenGL

Inizializzazione per il depth test

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

Aggiornamento di lightProjectionMatrix

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*

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);
```

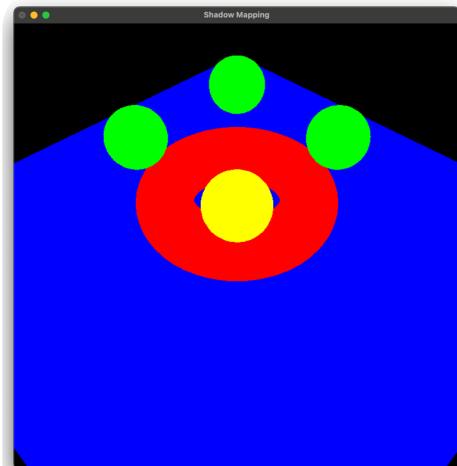
Codice della funzione UpdateEyePosition()

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

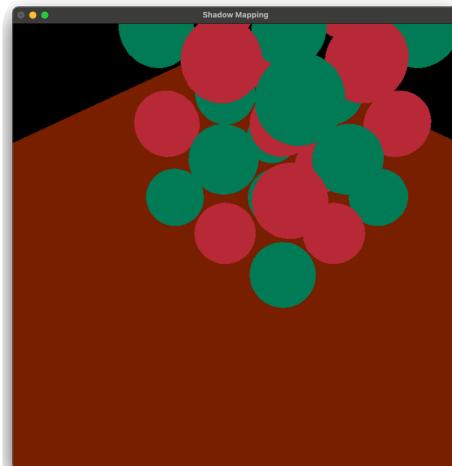
```
glBindTexture(GL_TEXTURE_2D, shadowMapTexture);
glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, 0, 0,
shadowMapSize, shadowMapSize, 0);
```

Generazione della shadow map

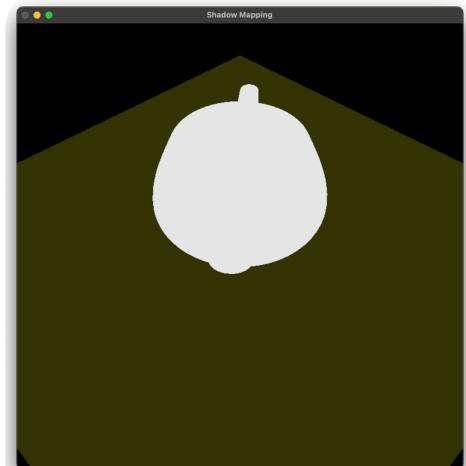
FirstStep() - Risultati



Scena 1



Scena 2



Scena 3

Generazione della shadow map

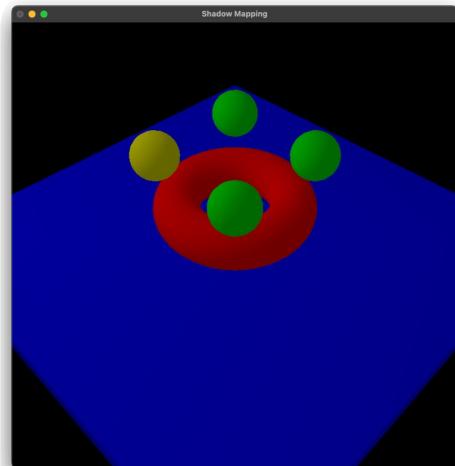
SecondStep() - Codice

```
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);
```

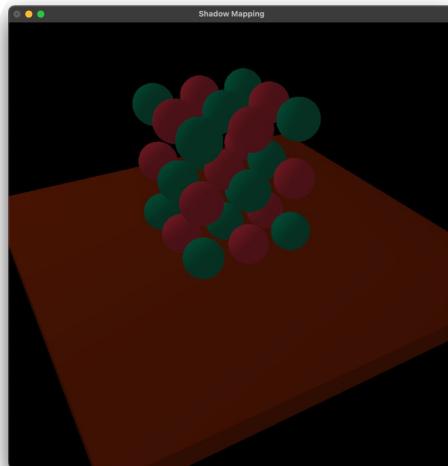
Codice della funzione SecondStep()

Generazione della shadow map

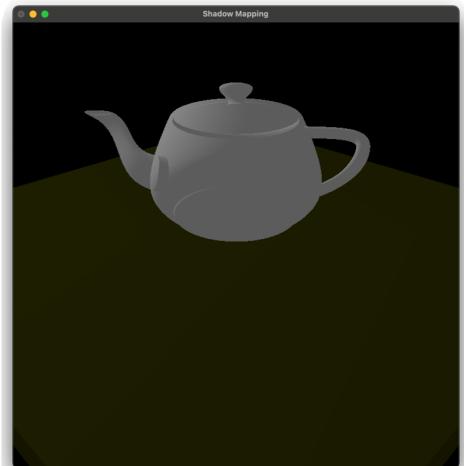
SecondStep() - Risultati



Scena 1



Scena 2



Scena 3

Applicazione del depth test

ThirdStep() - Codice

```
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);
```

Applicazione del depth test

ThirdStep() - Codice

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

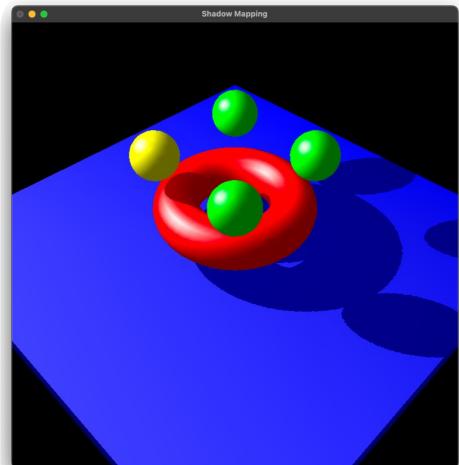
```
glTexParameteri(  
    GL_TEXTURE_2D,  
    GL_TEXTURE_COMPARE_FUNC,  
    GL_LEQUAL );
```

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

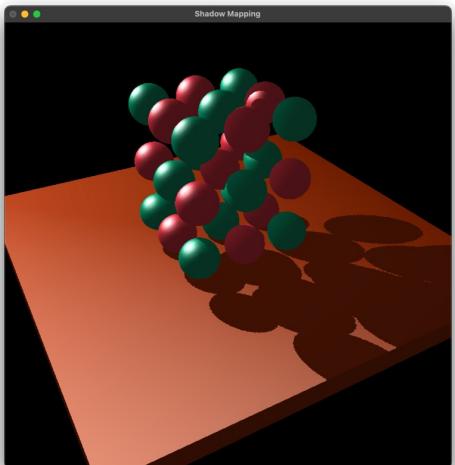
```
glAlphaFunc(GL_GEQUAL, 0.99f);  
 glEnable(GL_ALPHA_TEST);  
 DrawScene(angle, scene);
```

Applicazione del depth test

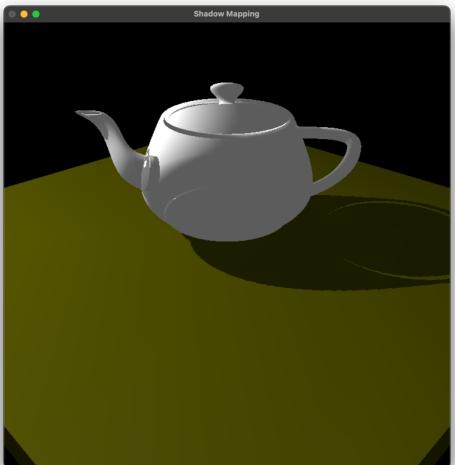
ThirdStep() - Risultato finale



Scena 1



Scena 2



Scena 3

Limitazioni dello spazio immagine

Field of view

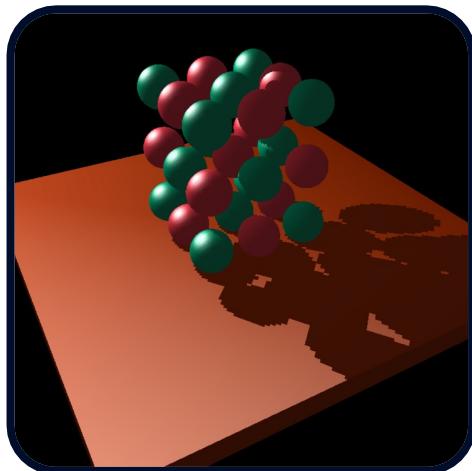
Multiple light sources

Aliasing

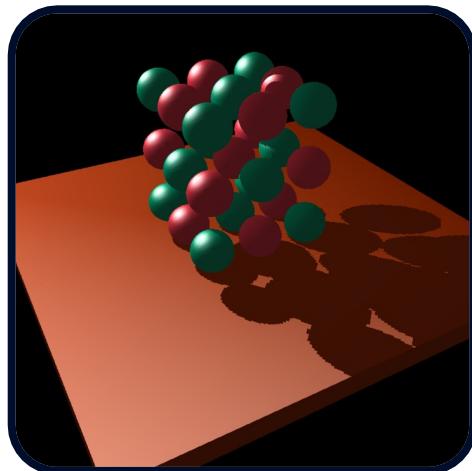
Self shadowing

Limitazioni dello spazio immagine

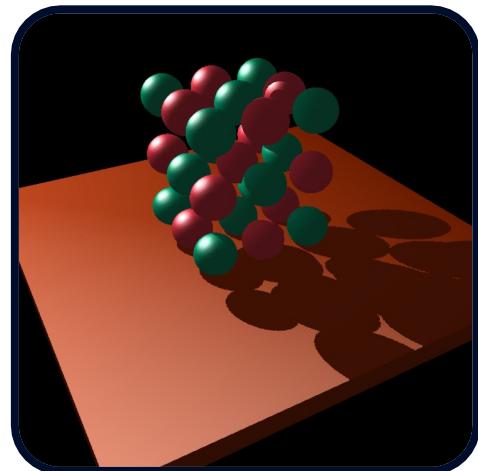
Aliasing



Risoluzione 128x128



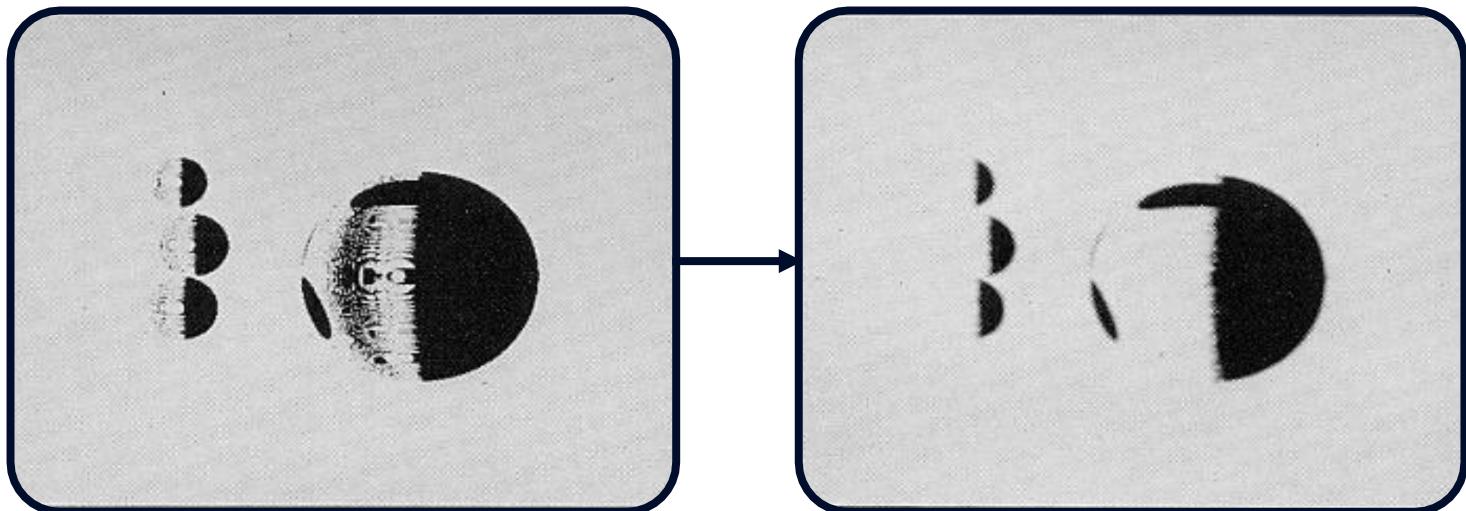
Risoluzione 256x256



Risoluzione 512x512

Limitazioni dello spazio immagine

Self shadowing



Applicazione del "biasing" della shadow line

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 **superficie curve**.

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%

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**.

Svantaggi

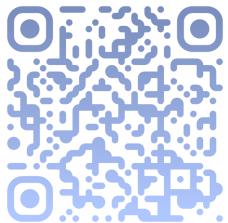
- Si possono presentare **problemi di aliasing** nell'utilizzare **texture a bassa risoluzione** per la **shadow map**.
- È necessario **renderizzare la scena per ogni fonte luminosa** e ottenere una **shadow map** diversa per ogni luce (**o in caso di luci puntiformi**).
- La fonte luminosa può **generare 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**.

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)**.*

Grazie!

Repository:



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

