



Shadow Mapping

▼ Per la presentazione

- Durata: max 25min.
- Effettuare una chiamata intermedia per capire se si sta svolgendo l'elaborato nel modo giusto.
- Capire come funziona lo shadow mapping anche su Blender.
- Implementare alcuni esempi utilizzando le librerie OpenGL e ed il linguaggio di programmazione GLSL (per lo shading) o altri.

GLSL è un linguaggio di programmazione che costruisce dei nuclei (kernel) che entrano nella pipeline grafica di OpenGL e ne alterano il comportamento.

- Scrivere un documento dove raccogliere tutte le informazioni a riguardo che, eventualmente, può essere consegnato.

▼ Traccia

Shadow mapping (or *projective shadowing*) is a process by which shadows are added to 3D computer graphics. Lance Williams introduced this concept in 1978 in a paper entitled "**Casting curved shadows on curved surfaces**".

Since then, it has been used in pre-rendered and real-time scenes in many console and PC games.

Shadows are created by [testing whether a pixel is visible from the light source](#) by comparing it to a *z-buffer* or depth image of the light source's view, stored in the form of a texture.

▼ Casting curved shadows on curved surfaces

▼ Introduzione

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

- **MIGLIORARE L'INTELLIGIBILITÀ DI UNA SCENA**, cioè la chiarezza dell'interpretazione (es. microscopio elettronico, viste aeree, ...).
- **COMPRENSIBILITÀ DI UN OGGETTO** e la sua posizione relativa.
- **IL LIVELLO DI REALISMO** di un'animazione.

Storicamente, sono stati pubblicati diversi metodi per la determinazione delle ombre di oggetti in una scena digitale. Ad oggi (1978), tali algoritmi si restringono al solo campo di determinazione delle ombre in scene composte unicamente di poligoni planari.

Un poligono planare è una forma geometrica i cui punti risiedono tutti sullo stesso piano. È anche detto superficie piatta.

▼ L'algoritmo proposto da L. Williams (1978)

Ecco che, quindi, in questo articolo accademico si introduce un algoritmo che utilizza il calcolo delle superfici visibili (tramite Z -buffer) per visualizzare le ombre generate da superfici curve.

Una superficie curva è una forma geometrica su due dimensioni. È l'opposto di una superficie piatta.

Nella computer grafica, per renderizzare una superficie curva sono necessari tanti piccoli poligoni planari (patches). Maggiore è il numero di poligoni, maggiore sarà la levigatezza della curva.

Il costo della determinazione delle ombre per ogni fonte luminosa è circa il doppio del costo del rendering della scena senza ombre, a cui bisogna aggiungere un piccolo overhead che dipende dalla sola risoluzione dell'immagine.

Per ottenere buoni risultati, è necessario porre la propria attenzione su tutti gli aspetti del calcolo delle ombre delle superfici visibili:

- **DITHER** (tremolio).
- **INTERPOLATION** (interpolazione).
- **GEOMETRIC QUANTIZATION** (quantizzazione).
- **SELF-SHADOWING**

È una caratteristica per cui un oggetto in movimento può generare la propria ombra su sè stesso o su altri oggetti nei dintorni.

▼ Calcolo della visibilità tramite Z -buffer

▼ L'algoritmo di E. Catmull (1974)

Nel paper "A Subdivision Algorithm for Computer Display of Curved Surfaces" (si veda [R1]), Catmull presenta un metodo per **produrre superfici curve ombreggiate utilizzando "patches" tridimensionali curve**, in contrasto con il precedente metodo che fa uso di patch poligonali.

In generale, questo metodo è poco efficiente sia per il tempo che impiega che per la memoria, in quanto richiede un gran numero di suddivisioni delle patches per arrivare ad ottenere una patch della dimensione di un "raster element" (il più piccolo elemento rappresentabile a schermo, e.g. pixel). Tuttavia, esiste una classe di patch - *le patch bicubiche* - che possono essere suddivise molto velocemente.

Inoltre, è possibile mappare delle fotografie sulle patch in modo tale da applicare delle texture alle immagini generate dal computer.

Un "frame buffer", è un'area di memoria (buffer) utilizzata per conservare un'intera immagine digitale (frame).

Il frame buffer può essere utilizzato come intermediario tra il processore ed il video driver, per supportare il suo lavoro di continuo refresh dello schermo.

Lo Z -buffer è un'area di memoria (buffer) utilizzata per stabilire quali sono le superfici visibili di una scena, conservando il valore z (profondità) di ogni punto dell'immagine.

Nella fase di rendering, per determinare la visibilità degli oggetti, si confrontano i loro valori z con quelli presenti nel buffer. Questo algoritmo è discretamente buono per tutti i tipi di scene, in quanto richiede solo una misura di confronto della profondità tra gli oggetti che devono essere visualizzati. Inoltre, non è necessario alcun tipo di pre-ordinamento, quindi è possibile gestire facilmente anche scene molto complesse.

- **A LIVELLO DEI PIXEL**, il Z -buffer è ordinato implicitamente tramite algoritmo di "radix sort" in X e Y con indicizzazione in Z .
- **SULLE DIMENSIONI X e Y** , l'ordinamento è dato dall'algoritmo di "bucket sort".
- **SULLA DIMENSIONE Z** , l'indice dell'ordinamento è 1, cioè è necessario **un singolo confronto per ogni oggetto**.

Il calcolo della visibilità tramite Z -buffer, quindi, presenta 3 vantaggi chiave:

1. **Può essere utilizzato per scene indefinitamente grandi.**
2. **Il costo cresce linearmente rispetto al numero di oggetti da ordinare nella profondità della scena.**
3. **Associa all'immagine finale una "depth map" della scena, molto utile per un'eventuale fase di post-produzione.** Il suo costo non dipende dalla complessità della scena, ma solo dalla risoluzione dell'immagine.

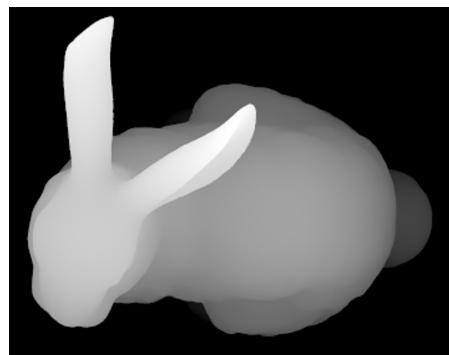
L'algoritmo che si vuole introdurre in questo articolo accademico tenta proprio di sfruttare le informazioni offerte dalla depth map restituita dal calcolo della visibilità tramite Z -buffer.

▼ Depth map

https://en.wikipedia.org/wiki/Depth_map

Una mappa di profondità è un'immagine che contiene la misura della distanza di una superficie di un oggetto nella scena da un punto di vista. Il termine "depth map" è spesso analogo ai termini "depth buffer" o Z -buffer (dove Z , per convenzione, è l'asse di profondità dello spazio dell'osservatore).

La mappa di profondità si costruisce assegnando ad ogni suo pixel un valore rappresentante la distanza di tale pixel dal punto di riferimento dell'osservatore. Ad esempio, in questo paper si utilizzano valori a 16 bit.



<https://lookingglassfactory.com/blog/depth-map>

▼ Quali informazioni offrono le ombre?

La generazione delle ombre da una fonte luminosa puntiforme su una superficie piatta rappresenta una proiezione della scena stessa su un piano, proprio come per le trasformazioni prospettiche.

Una scena renderizzata con l'utilizzo delle ombre, quindi, contiene due viste in una sola immagine.

Ovviamente, in generale, le ombre non vengono diffuse solo su superfici planari, ma potrebbero presentarsi su un qualsiasi tipo di superficie.

In questo caso, due viste sono comunque sufficienti. Esse, però, devono utilizzare il calcolo della visibilità con Z-buffer.

▼ Descrizione dell'algoritmo

Le due viste sono costruite nel modo seguente:

1. **La prima vista della scena si costruisce dal punto di vista della fonte luminosa.** Si memorizzano solo i valori z e non quelli di shading.
2. **L'altra vista della scena si costruisce dal punto di vista dell'osservatore.** È necessaria una trasformazione lineare che mappi i punti di coordinate (X, Y, Z) dello spazio di vista dall'osservatore nei punti di coordinate (X', Y', Z') dello spazio di vista dalla fonte luminosa.

Utilizzando questa trasformazione lineare:

- Ogni punto generato nella vista dell'osservatore viene trasformato in un punto della vista della fonte luminosa.
 - Si calcola la visibilità nel Z -buffer prima di generare le ombre (computing its shading value). Se il punto non è visibile, vuol dire che è già in ombra e, quindi, deve essere ombreggiato (shaded) di conseguenza.
- In particolare, questa parte dell'applicazione della trasformazione lineare viene intrapresa come un lavoro di post-produzione. In questo modo, l'algoritmo:
- **Scurisce le luci della scena** se sono nascoste dal punto di vista della fonte luminosa, anche se non dovrebbero apparire.
 - **Soffre maggiormente dei problemi legati alla quantizzazione**, in base alla risoluzione del Z -buffer.
 - **Presenta delle prestazioni efficienti**, in quanto non dipende dalla complessità della scena (numero di oggetti), ma solo dalla risoluzione dell'immagine (e.g. si applica la trasformazione lineare per ogni punto visibile nell'immagine finale).

▼ Limitazioni dello spazio immagine

La generalizzazione alle superfici curve e il costo lineare sono le principali caratteristiche che distinguono l'algoritmo proposto.

Queste caratteristiche sono dovute al fatto che tutti i calcoli sono eseguiti in uno spazio immagine, cioè nello spazio bidimensionale visibile a schermo.

Questo approccio, tuttavia, porta con sè alcune limitazioni, da analizzare con attenzione e mettere a confronto con i vantaggi dell'algoritmo.

- **FIELD OF VIEW**

Questo algoritmo, come già detto in precedenza, si basa due viste: la vista dell'osservatore e la vista della fonte luminosa.

È fondamentale garantire che tutti gli oggetti in grado di generare un'ombra nel volume di vista dell'osservatore siano all'interno del volume di vista della fonte luminosa.

Infatti:

- I punti trasformati nella vista della fonte luminosa che giacciono al di fuori di questo volume di vista sono, comunque, illuminati.
- Al contrario, le ombre devono essere calcolate solo per quei punti trasformati che risiedono all'interno del volume di vista della fonte luminosa.

- **LIGHT SOURCE LOCATION**

Non è sempre vero che la fonte luminosa debba trovarsi al di fuori del volume di vista dell'osservatore.

La fonte luminosa può generare delle ombre solo all'interno del volume di vista dell'osservatore.

Nel caso in cui la fonte luminosa si trovi proprio all'interno del volume di vista dell'osservatore, quest'ultima può generare delle ombre in tutte le direzioni; pertanto, la sfera di illuminazione deve essere sezionata in un molteplice numero di viste [R5].

Il calcolo della visibilità tramite Z -buffer per tutte queste viste non è sensibilmente più costoso rispetto al calcolare la visibilità in una vista singola.

Il vero problema si ha nel calcolare la trasformazione lineare dai punti nello spazio dell'osservatore, ai punti in ogni sezione della fonte luminosa.

La principale difficoltà di questo metodo è la maggior quantità di memoria richiesta.

- **QUANTIZATION e ALIASING**

La garanzia di avere una rigida prospettiva, sia dal punto di vista dell'osservatore che dal punto di vista di una fonte luminosa particolarmente vicina alla scena, può aumentare i problemi di quantizzazione legati alla trasformazione da uno spazio immagine all'altro.

Infatti, la quantizzazione e l'aliasing sono i principali ostacoli degli algoritmi che lavorano sullo spazio immagine (si vedano [R1], [R6] e [R7]).

- **SELF-SHADOWING**

La trasformazione di un punto dallo spazio dell'osservatore allo spazio della fonte luminosa dovrebbe, teoricamente, terminare esattamente sulla superficie dello spazio immagine.

Tuttavia, a causa delle approssimazioni dell'aritmetica del calcolatore e della quantizzazione delle superfici Z -buffer, il punto trasformato potrebbe trovarsi leggermente sopra o sotto la superficie dello spazio immagine.

Siccome l'algoritmo richiede che un punto P da illuminare si trovi su una superficie visibile, allora si sottrae un bias dal valore z del punto P nel Z -buffer dopo la sua trasformazione (in generale, tale bias è integrato proprio nella trasformazione lineare).

Il bias ha la capacità di muovere leggermente la "shadow line", in modo tale da:

- Risolvere i problemi di quantizzazione
- Impedire alle superfici che sono chiaramente visibili alla fonte luminosa di oscurarsi.

- **SELF-SHADOWING (cont.)**

Una superficie curva deve oscurarsi solo alla fine, a mano a mano che la superficie stessa si allontana dalla fonte luminosa.

Per ottenere questo effetto di passaggio da piena luce a oscurità, si può utilizzare il bit meno significativo del valore z nel Z -buffer.

Tuttavia:

L'intensità dell'ombra può cambiare rapidamente quando l'errore di quantizzazione batte con la griglia di campionamento, producendo così un effetto moiré (interferenza).

▼ Applicazione dell'algoritmo

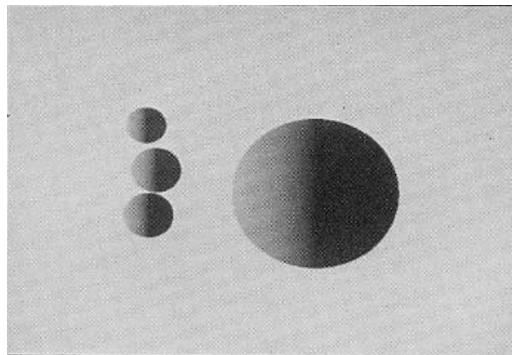


Figura 1. Quattro sfere dal punto di vista dell'osservatore.

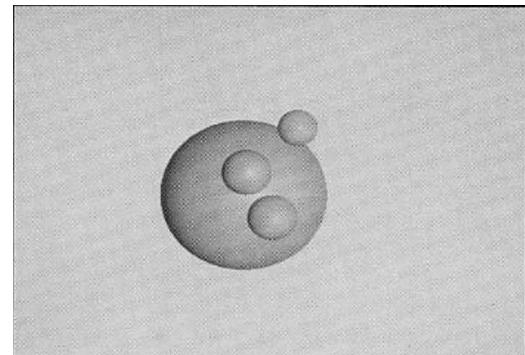


Figura 2. Le quattro sfere dal punto di vista della fonte luminosa.

Le figure 1, 2 mostrano una semplice scena dal punto di vista dell'osservatore e dal punto di vista della fonte luminosa, rispettivamente.

In particolare, la fonte luminosa è:

- Posizionata all'infinito.
- Ruotata di 100° rispetto all'asse verticale (posizionato a sinistra dell'osservatore).

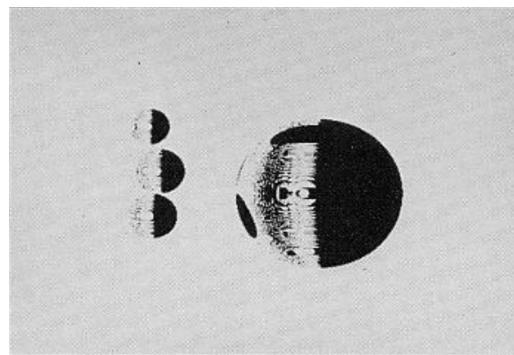


Figura 3. Calcolo delle ombre con un bias minimo per mostrare l'effetto moiré dovuto alla quantizzazione.

La figura 3 svela l'effetto moiré dovuto alla quantizzazione. Si può ottenere questo risultato utilizzando un bias, da sottrarre ai valori z , molto piccolo per la trasformazione lineare dei punti dallo spazio dell'osservatore allo spazio della fonte luminosa.

Da notare che l'errore di quantizzazione è maggiore in due aree: al limite della "shadow line", e in una fascia alla sua sinistra sulla sfera più grande.

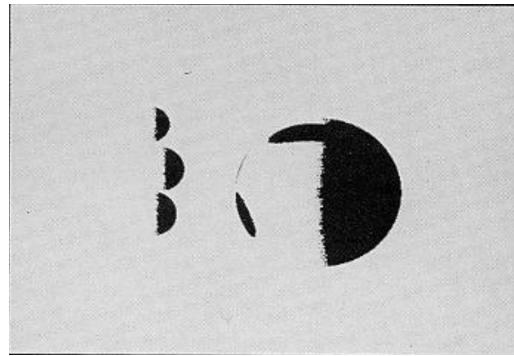


Figura 4. Calcolo delle ombre con un bias maggiore e tecnica del tremolio (dither) applicata.

Per migliorare ulteriormente la qualità del calcolo delle ombre:

- Bandlimiting the Z partition (?)
- Si sceglie di trattare questo tipo di rumore come un problema di quantizzazione piuttosto che un problema di aliasing.

Il segnale di errore in un sistema di quantizzazione è fortemente legato al segnale stesso. Quindi, l'aggiunta di un rumore a valori aleatori nell'intervallo di un singolo quanto annulla questa correlazione e riduce, di conseguenza, l'effetto moiré.

La figura 4 illustra il calcolo delle ombre utilizzando:

- Un bias negativo più grande.
- Un'interpolazione bilineare dei valori z della fonte luminosa alle coordinate X, Y dei punti dell'osservatore trasformati, che sono stati, inoltre, oscillati dall'aggiunta di valori aleatori con distribuzione gaussiana nell'intervallo $[-5, +5]$.

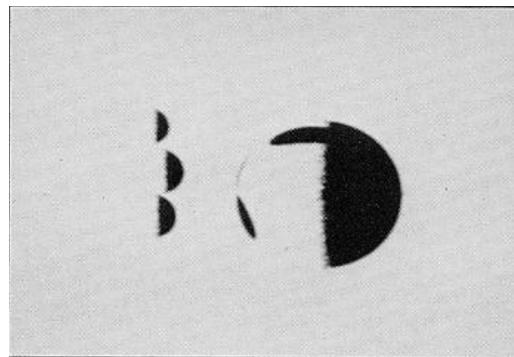


Figura 5. Applicazione di un filtro passa-basso.

Come ultimo passaggio, si cerca di smussare il passaggio da luce a ombra sulla "shadow line" utilizzando:

- Un "edge dequantizing filter" (si veda [R8]).
- Un filtro passa-basso.

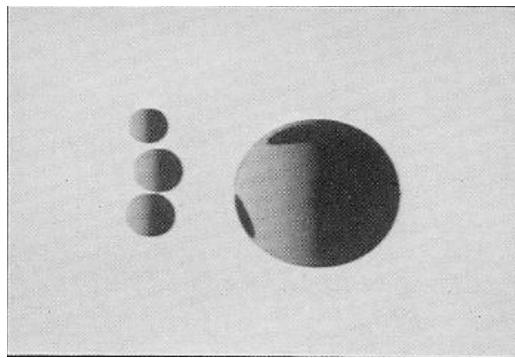


Figura 6. Applicazione delle ombre sulla figura 1.

Infine, la figura 6 mostra che il problema del self-shadowing non è particolarmente significativo in pratica. L'ombreggiatura delle sfere in base alla posizione della fonte luminosa genera una transizione dolce che nasconde l'errore di quantizzazione: le cosiddette "translucent shadows".

Le ombre traslucide spesso sono migliori delle ombre scure, in quanto si aggiungono meglio all'ambiente da simulare.

Inoltre, **l'applicazione del filtro passa-basso prima della realizzazione dell'immagine finale approssima (o simula) quell'effetto di penombra dato dalle fonte di luce reali**.

▼ Conclusioni

- L'algoritmo opera con successo su scene in cui vi sono superfici curve.
- L'algoritmo è pensato per sfruttare il parallelismo temporale (pipeline) dei processori, soprattutto per la fase di trasformazione delle coordinate.
- L'algoritmo mostrato è solo un esempio di una più grande classe di algoritmi estremamente generici che presentano una proprietà di aumento dei costi molto desiderabile.
- Il costo dell'algoritmo aumenta in modo lineare rispetto alla complessità dell'ambiente. Tale costo è circa il doppio del rendering della scena normale (perché non è richiesto alcun calcolo di shading), a cui aggiungere il costo della trasformazione dei punti dallo spazio dell'osservatore allo spazio della fonte luminosa.
- La trasformazione lineare viene eseguita strettamente come un'operazione di post-produzione. Pertanto, il costo dipende solo dalla risoluzione dello schermo / spazio immagine (che corrisponde ad una scena la cui profondità è 1).

▼ Futuri sviluppi

- Si può pensare di utilizzare un hardware specifico per migliorare la velocità dell'esecuzione (es. dispositivi digitali in grado di eseguire velocemente una moltiplicazione tra matrici omogenee di trasformazione 4×4).
- La complessità del software necessario per implementare l'algoritmo è minima se si ha a disposizione la memoria necessaria.

▼ Riferimenti

[P1] Casting curved shadows on curved surfaces

Casting curved shadows on curved surfaces | Proceedings of the 5th annual conference on Computer Graphics Lab, New York Institute of Technology, Old Westbury, New York
 <https://dl.acm.org/doi/abs/10.1145/800248.807402>

[R1] A Subdivision Algorithm for Computer Display of Curved Surfaces

<https://apps.dtic.mil/sti/pdfs/ADA004968.pdf>

[R2] A Solution to the Hidden Surface Problem

<https://dl.acm.org/doi/pdf/10.1145/280811.280918>

[R4] Computer determination of depth maps

[https://doi.org/10.1016/0146-664X\(73\)90024-5](https://doi.org/10.1016/0146-664X(73)90024-5)

[R5] Shadow algorithms for computer graphics

Shadow algorithms for computer graphics | ACM SIGGRAPH Computer Graphics

Shadows are advocated for improved comprehension and enhanced realism in computer-synthesized images. A classification of shadow algorithms delineates three approaches: shadow computation during scanout; division of object surfaces into shadowed and ...

 <https://dl.acm.org/doi/abs/10.1145/965141.563901>

[R6] A scan line algorithm for computer display of curved surfaces

A scan line algorithm for computer display of curved surfaces | ACM SIGGRAPH Computer Graphics

The conventional procedure for generating shaded images of curved surfaces is to approximate each surface element by a mosaic of polygons and to then apply one of several established polygon display algorithms. The method described here produces an ...

 <https://dl.acm.org/doi/abs/10.1145/988437.988440>

[R7] The Aliasing Problem in Computer-Synthesized Shaded Images

<https://apps.dtic.mil/sti/pdfs/ADA038979.pdf>

[R8] Computer Processing of Line-Drawing Images

Computer Processing of Line-Drawing Images | ACM Computing Surveys

New York University West 177th Street & Harlem River, Bronx, New York

 <https://dl.acm.org/doi/abs/10.1145/356625.356627>



[R9] Models of light reflection for computer synthesized pictures

Models of light reflection for computer synthesized pictures | Proceedings of the 4th annual conference

You will be notified whenever a record that you have chosen has been cited.

 <https://dl.acm.org/doi/abs/10.1145/563858.563893>

[R10] A head-mounted three dimensional display

A head-mounted three dimensional display | Proceedings of the December 9-11, 1968, fall joint computer conference

You will be notified whenever a record that you have chosen has been cited.

 <https://dl.acm.org/doi/abs/10.1145/1476589.1476686>

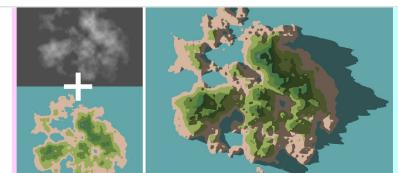
▼ Altri riferimenti

Realtime shadow casting on 2D terrain

Become a member!

<https://www.youtube.com/@BarneyCodes/join>

 <https://youtu.be/bMTeCqNkld8>



What are object space and image space in computer graphics?

Answer (1 of 2): Differences are listed as: Object Space Method a. Deals with object definition directly. b. It compares the objects and part of objects to each other within the scene definition to determine which surfaces, as a whole to be labeled as visible. c. Initiated for vector graphics...

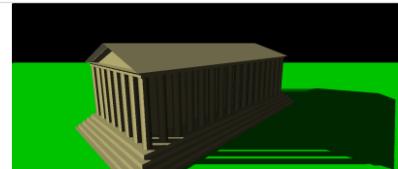
 <https://www.quora.com/What-are-object-space-and-image-space-in-computer-graphics>



Shadow mapping

Shadow mapping or projective shadowing is a process with which shadows are calculated and rendered in computer graphics. The concept was introduced by Lance Williams in 1978, in a paper entitled "Casting curved shadows on curved surfaces." Since then, it has been used both in pre-rendered

 https://it.wikipedia.org/wiki/Shadow_mapping



<https://web.archive.org/web/20040619120702/http://developer.nvidia.com/attach/6769>

[P2] Rendering antialiased shadows with depth maps

Rendering antialiased shadows with depth maps | Proceedings of the 14th annual conference on Computer graphics and interactive techniques

You will be notified whenever a record that you have chosen has been cited.

 <https://dl.acm.org/doi/abs/10.1145/37401.37435>

[P3] Fast Shadows and Lighting Effects Using Texture Mapping

<https://dl.acm.org/doi/pdf/10.1145/133994.134071>

OpenGL Shadow Mapping Tutorial - Paul's Projects

Shadow mapping was introduced by Lance Williams in 1978, in a paper entitled "Casting curved shadows on curved surfaces".

It has been extensively used since, both in offline rendering and real time graphics.

Shadow mapping is used by Pixar's Renderman and was used on major films such as "Toy Story".

 <https://www.paulsprojects.net/tutorials/smt/smt.html>