Shadow Algorithms for Walktrough Applications

Yiorgos Chrysanthou

University College London

Outline

- Introduction
- Sharp shadows
- Soft shadows
- Conclusion

Why Use Shadows

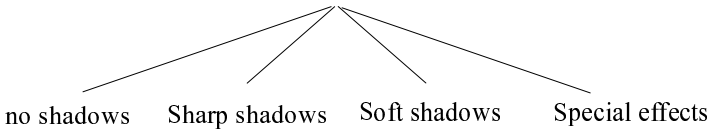
- They add to the realism of a computer generated image
  - Our visual perception is very sensitive to shadows
- They provide information for the spatial relationships between objects

Shadows are Complex

- In the real world sources of light are not points
- The intensity within a shadow is not constant
  - umbra, the part that sees nothing of the source
  - penumbra, part that receives some light
- In computer graphics we simplify and cheat

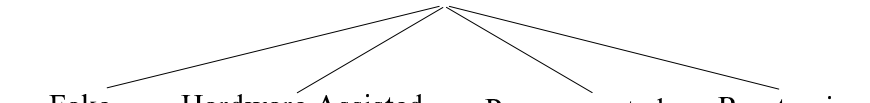
Current shadow Methods

- There exist a very large number of methods
- We are interested in methods suitable for interactive walkthroughs, speed is crucial
- We will classify them on complexity:



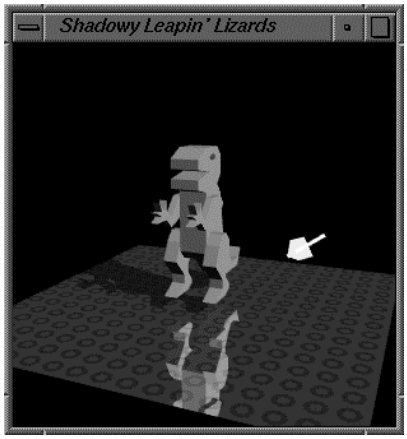
Sharp Shadows

- Source is assumed to be a point or direction



Fake	Hardware Assisted	Pre-computed	Ray tracing
Fast but often inadequate • projection	Use of specialized hardware to generate shadows • Shadow Z-buffer • Shadow volumes	Shadows are pre-computed and stored for repeated use • SVBSP • Shadow Tiling	It's still very slow for large scenes

Fake shadows: projection on ground



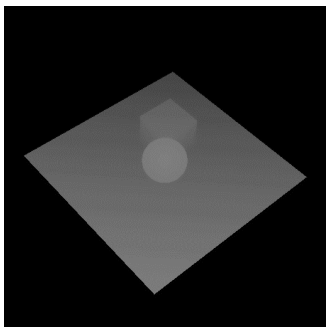
(image from openGL demo)

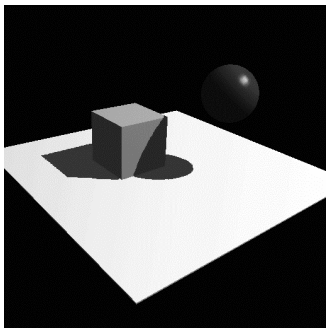
- Objects are compressed using a matrix transformation and pasted to the ground [Blinn 88]
- No inter-object shadows
- Very fast

Shadow Z-buffer

- Compute a Z-buffer from the source
  - use the light source as a view point and render the objects to get the depth information (shadow Z-buffer)
- Run a normal Z-buffer with shadow calculation
  - from the view point, each point in this buffer is mapped to the shadow buffer, if the Z value is equal to that stored there then the point is lit, otherwise is in shadow

Shadow Z-buffer with openGL





(images taken from ????)

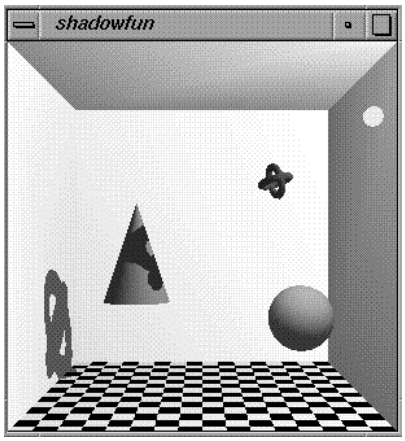
Image from source Resulting shadows

- This technigue can be accelerated by using texture mapping hardware [Segal 92]

Shadow Volume Method

- Shadow volume (SV) is the volume of space below a polygon that cannot see the source (a culled pyramid)
- During rendering of image, the line from a point visible through a pixel to the eye is intersected with all object SVs
- The number of intersections indicates if the point is in shadow or not [Crow 77]

Shadow Volumes with openGL



(image from an SGI demo)

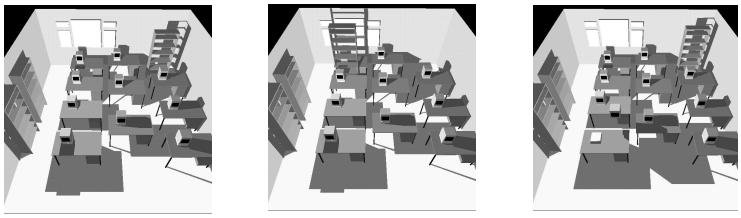
- Shadow volumes are rendered at each frame
- The stencil buffer is used for counting how many SV are crossed [Heidmann 91]
- Sometimes not all objects are used for casting shadows

Shadow Volume BSP tree

- A BSP tree is build incrementally using the shadow planes
- Polygons are added to the tree
  - if they fall in an ***IN*** region (behind a set of shadow planes) they are shadowed
  - otherwise they are lit and their shadow planes are used to expand the tree
- [Chin 89]

SVBSP trees and Shadow Tiling

- Shadows are stored as detail polygons on top of scene polygons
- Shadows just rendered at run time
- Everything is precomputed but small changes are possible



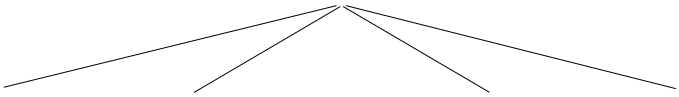
(images taken from Chrysanthou 95)

Soft Shadows

- Source has a finite extend
- Images look a lot more realistic

(Image taken from Nishita and Nakamae)

Soft Shadows



Hardware Assisted

Mainly treat the light source as a collection of points

- Accumulation buffer
- Shadow volumes
- Shadow textures

Pre-computed

Mainly analytical computation on the geometry of the source

- SVBSP
- Discontinuity Meshing

Radiosity

This is also pre-computed

- Hemi-cube
- Ray casing

Ray tracing

- Distributed ray tracing
- Cone Tracing

Hardware assisted

- Most of these techniques are extensions of sharp shadow methods
  - [Haeberlei 90] Accumulation buffer for the shadow Z-buffer
  - [Brotman 84] uses shadow volumes
- These are mainly approximate solutions treating the source as a collection of points

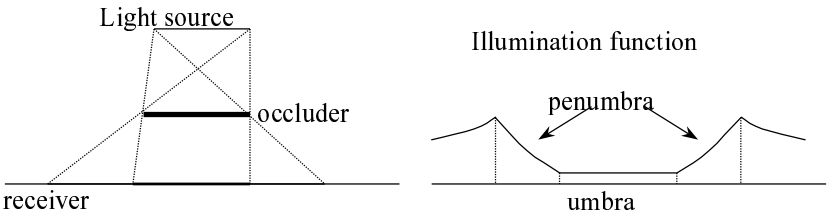
Shadow Textures

- Shadows are quickly computed and stored as textures on the receiving polygons
- Displayed using hardware in real-time
- Two example methods:
  - [Heckbert 97] source is sampled and results combined
  - [Cyril 98] uses convolution

SVBSP Trees

- Idea is similar to the SVBSP for point sources
- Two trees are built, one using the umbra planes and one using the penumbra ones
- Scene polygons are partitioned into lit, umbra and penumbra pieces

Discontinuity Meshing

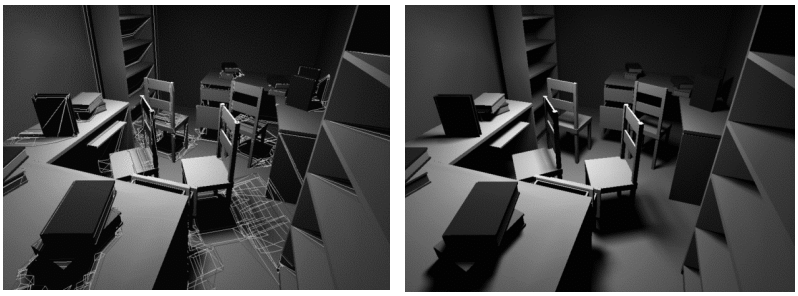


- Subdivide at discontinuity points
- Compute illumination intensity at discontinuity points
- Quadratic approximation on segments between discontinuity points

Discontinuity Meshing

- Very high quality shadows
- Slow and prone to floating point errors

(images taken from Dretakis)



Radiosity

- Scene polygons are subdivided into a mesh, or the illumination is stored as a texture
- Very realistic results
- Good for static scenes but not for moving objects

Conclusion

- A very large number of shadow algorithms exist
- Most of them are unsuitable for walkthroughs of very complex scenes:
  - with pre-computation methods scene cannot be modified
  - on-the-fly methods are not fast enough yet
- Only very limited solutions are currently possible, such as fake shadows