Image Processing and Computer Graphics

Shadow Algorithms

Matthias Teschner

Computer Science Department
University of Freiburg
Outline

- introduction
- projection shadows
- shadow maps
- shadow volumes
- conclusion
Motivation

- shadows help to
  - improve the realism in rendered images
  - illustrate spatial relations between objects
Goal

- determination of shadowed parts in 3D scenes

- only the geometry is considered

[Akenine-Moeller et al.: Real-time Rendering]
Context

- Shadow algorithms are not standard functionality of the rasterization-based rendering pipeline.
- Rendering pipeline:
  - Generates 2D images from 3D scenes (camera, light, objects).
  - Evaluates lighting models using local information.
  - Spatial relations among objects are not considered.

Diagram:
- Camera
- Image plane/frame buffer
- View volume/frustum
- Light
- Occlusion of the light source is not considered.
Outline

- introduction
- projection shadows
- shadow maps
- shadow volumes
- conclusion
Projection Shadows

- project the primitives of occluders onto a receiver plane (ground or wall) based on the light source location
- projected primitives form the shadow geometry
- projection matrix

\[ P = l n^T - (n \cdot l) I_4 \]

\[ n = \{ax + by + cz + d = 0\} \]
\[ = (a, b, c, d)^T \]

\[ p = (p_x, p_y, p_z 1)^T \]

light source position

receiver

occluder
Projection Shadows
Implementation

- draw the receiver plane
  - increment stencil where the receiver is rendered
- disable depth test
- draw shadow geometry (projected occluders) for stencil=1
- enable depth test
- draw occluders
Projection Shadows

Issues

- restricted to planar receivers
- no self-shadowing
- antishadows

[Akenine-Moeller et al.: Real-time Rendering]
**Outline**

- introduction
- projection shadows
- shadow maps
- shadow volumes
- conclusion
Concept

- see shadow casting as a visibility problem
- scene points are
  - visible from the light source (illuminated)
  - invisible from the light source (in shadow)
- resolving visibility is standard functionality in the rendering pipeline (z-buffer algorithm)
Algorithm

- render scene from the light source
- store all distances to visible (illuminated) scene points in a shadow map
- render scene from the camera
- compare the distance of rendered scene points to the light with stored values in the shadow map
- if both distances are equal, the rendered scene point is illuminated
**Coordinate Systems**

- in the second rendering pass, for each fragment, its position in the shadow map has to be determined
  - convert the 3D position of a fragment to object space
  - apply modelview transform of the light $L^{-1}M$
  - apply projective transform of the light $P_{\text{light}}$
  - homogenization and screen mapping
  - mapping to texture space
  - results in $(x', y', z', 1)^T$
  - $z' > \text{shadowMap}(x', y') \rightarrow \text{point in shadow}$
Shadow Map Generation

scene is rendered from the position of the light source

shadow map. a texture that represents distances of illuminated surface points to the light source.

[Akenine-Moeller et al.: Real-time Rendering]
Scene Rendering

in the second rendering pass, \( v_a \) and \( v_b \) are rendered. \( v_a \) is represented in the shadow map (illuminated). \( v_b \) is occluded and not represented in the shadow map (in shadow).

in a second rendering pass, the camera view is generated. For each fragment, it is tested whether it is represented in the shadow map or not.

[Akenine-Moeller et al.: Real-time Rendering]

University of Freiburg – Computer Science Department – Computer Graphics - 15
**Algorithm**

- use two depth buffers
  - the “usual” depth buffer for the view point
  - a second depth buffer for the light position (shadow map)
- render the scene from the light position into the shadow map
- render the scene from the view position into the depth buffer
  - transform depth buffer values to shadow map values
  - compare transformed depth buffer values and shadow map values to decide whether a fragment is shadowed or not
Aliasing

- discretized representation of depth values can cause an erroneous classification of scene points
- offset of shadow map values reduces aliasing artifacts

[Mark J. Kilgard]
Sampling

- in large scenes, sampling artifacts can occur
- uniform sampling of the shadow map can result in non-uniform resolution for shadows in a scene
- shadow map resolution tends to be too coarse for nearby objects and too high for distant objects

[Stamminger, Drettakis]
Sampling
Perspective Shadow Maps

- scene and light are transformed using the projective transformation of the camera
- compute the shadow map in post-perspective space

[Stamminger, Drettakis]
Sampling
Perspective Shadow Maps

uniform shadow map

perspective shadow map [Stamminger, Drettakis]

University of Freiburg – Computer Science Department – Computer Graphics - 20
Summary

- image-space technique with two rendering passes
- no knowledge of the scene geometry is required
- works best for distant spot light sources
- light looks at the scene through a single view frustum
  - scene geometry outside the view frustum is not handled
- aliasing artifacts and sampling issues
Outline

- introduction
- projection shadows
- shadow maps
- shadow volumes
- conclusion
**Concept**

- employ a polygonal representation of the shadow volume
- point-in-volume test

Diagram:
- A is illuminated, ray does not intersect the shadow volume
- B is in shadow, ray enters the shadow volume
- C is illuminated, ray enters and leaves the shadow volume

Legend:
- light
- occluder
- receiver
- camera
Implementation Issues

- classification of shadow volume polygons into front and back faces
  - rays enter the volume at front faces / leave it at back faces
- stencil buffer values count the number of intersected front and back faces
Algorithm (Z-pass)

- render scene to initialize depth buffer
  - depth values indicate the closest visible fragment
- stencil enter / leave counting approach
  - render shadow volume twice using face culling
    - render front faces and increment stencil when depth test passes (count occluding front faces)
    - render back faces and decrement stencil when depth test passes (count occluding back faces)
  - do not update depth and color
- finally,
  - if pixel is in shadow, stencil is non-zero
  - if pixel is illuminated, stencil is zero
Implementation

- render the scene with only ambient lighting
- render front facing shadow volume polygons (without depth and color update) to determine how many front face polygons are in front of the depth buffer pixels
- render back facing shadow volume polygons (without depth and color update) to determine how many back face polygons are in front of the depth buffer pixels
- render the scene with full shading where stencil is zero
Example

Stencil value = +1+1+1-1-1-1 = 0
Example

Stencil value = +1 + 1 + 1 - 1 = 2
Example

- **light**
- **occluder**
- **zero**
- **+1**
- **+2**
- **+2**
- **+3**
- **camera**
- **illuminated point**
- **stencil value = 0**

University of Freiburg – Computer Science Department – Computer Graphics - 29
Example

stencil value = 1
stencil value = 0
Missed Intersections

**Missed** shadow volume intersection due to near plane clipping
Solutions

- Z-fail
  - counts the difference of occluded front and back faces
  - misses intersections behind the far plane
- Z-fail with depth clamping
  - do not clip primitives at the far plane
  - clamp the actual depth value to the far plane
- Z-fail with far plane at infinity
  - adapt the perspective projection matrix
Algorithm (Z-fail)

- render scene to initialize depth buffer
  - depth values indicate the closest visible fragment
- stencil enter / leave counting approach
  - render shadow volume twice using face culling
    - render back faces and increment stencil when depth test fails (count occluded back faces)
    - render front faces and decrement stencil when depth test fails (count occluded front faces)
  - do not update depth and color
- finally,
  - if pixel is in shadow, stencil is non-zero
  - if pixel is illuminated, stencil is zero
Improving Z-fail

- depth clamping
  - do not clip primitives to the far plane
  - draw primitives with a maximum depth value instead (clamp the actual depth value to the far plane)
- extend the shadow volume to infinity
- set the far plane to infinity (matrices in OpenGL form with negated values for n and f)

\[
\begin{pmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\
0 & 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\
0 & 0 & -1 & 0
\end{pmatrix} \quad f \to \infty \quad \Rightarrow \quad \begin{pmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\
0 & 0 & -1 & -2n \\
0 & 0 & -1 & 0
\end{pmatrix}
\]
Outline

- introduction
- projection shadows
- shadow maps
- shadow volumes
- conclusion
Summary

- projection shadows
  - restricted to planar receivers, no self-shadowing
- shadow maps
  - image-space technique, two rendering passes
  - works correct, if all relevant objects are "seen" by the light
  - sampling issues
- shadow volumes
  - requires a polygonal representation of the shadow volume
  - multiple rendering passes
  - clipping of shadow volume polygons has to be addressed
References

Shadow Maps


Shadow Volumes