Computer Graphics

- Shadow Algorithms -

Philipp Slusallek
Recap: Shadows

- Lightsource contributes to shading if the lightsource is visible from the sampling point
- Contributions of lightsources are added

```
Color shade( Point surfacePosition, Direction toCamera )
{
    Color outgoing = black;
    for ( light : lightsources )
    {
        if ( ! occluded( surfacePosition, light.position ) )
        {
            Color incoming = light.illuminate( surfacePosition );
            Direction toLight = light.position – surfacePoint;
            outgoing += brdf( incoming, toCamera, toLight );
        }
    }
    return totalColor;
}
```
Recap: Visibility Queries

- **Visibility queries**
  - Simplified ray tracing operations

- **Normal ray-scene intersection:**
  - Find first intersection with scene

- **Visibility Query:**
  - Find any intersection with scene (slightly faster)

- **Rasterization context**
  - Ray-scene intersection operation is not available
  - No access to other triangles from within the pipeline
Shadowing in Rasterization

- **Direct (subtractive) shadow rendering (historic)**
  - Render scene without shadows
  - "Add" shadows later by subtracting light (e.g. via geometry)
  - No consistent interpretation in terms of rendering equation

- **Shadow Volumes (Hard Shadows)**
  - Precompute volume representation of light visibility
  - Use visibility information during rendering

- **Shadow Maps (Hard Shadows)**
  - Precompute a discrete representation of light visibility
  - Use visibility information during rendering

- **Hybrid Rendering Techniques (Soft Shadows)**
  - Use cone-tracing in a scene approximation
SHADOW VOLUMES
Shadow Volumes

Point Light Source

Triangle

Shadow Volume
Shadow Volumes

Camera

back-faces away from camera count -1

front-faces to camera count 1
Shadow Volume Algorithm

1. fill z-buffer as seen from camera (disable writing to FB)
2. fill color buffer to black
3. for ( light : lightsources )
   4. initialize stencil buffer to zero
   5. compute all front facing shadow volume surfaces
   6. render to stencil buffer with value 1
   7. compute all back facing shadow volume surfaces
   8. render stencil buffer with value -1
   9. render scene from camera only, adding illum where shadow volume buffer != 0
Properties

- **Pro**
  - Per pixel accurate shadows without aliasing problems
  - Maps moderately well to hardware

- **Con**
  - Needs major changes to how scene is rendered

- **Performance**
  - Generate 3 faces per triangle-light combination
  - Poor performance for geometrically complex shadow casters
  - Very bandwidth intensive
Optimization: Silhouettes

- Mesh as seen from light source
- Only silhouette edges are relevant
Optimization: Silhouettes

Light Source

Silhouette edges

Camera
Finding silhouettes

- **Brute force approach (convex objects)**
  
  ```
  for ( light: scene )
    for ( edge : mesh )
    {
      normal1 = edge.face1.normal;
      normal2 = edge.face2.normal;
      camera = directionToCamera;
      direction1 := sign( dotProduct(normal1, camera) );
      direction2 := sign( dotProduct(normal2, camera) );
      isSilhouette[light, edge] := (direction1 != direction2);
    }
  ```

  - Finding general silhouettes is a difficult topic in itself

  - Possible optimization
    - Edges between planer triangles can be disregarded
Further Reading

Approximate approach:

Using Gauss Maps:
Shadow Volumes Example
SHADOW MAPS
Shadow Mapping

• **Idea:** Use Z-Buffer to store occlusion information

• **First pass:** Render scene from position of light source
  – For each pixel: Store distance from lightsource to object
  – Resulting data structure is called shadow map

• **Second pass:** Render scene from position of camera
  – For each pixel: Compare distance to lightsource with corresponding pixel in shadow map
Shadow Mapping

Light Source

Shadow Map
Shadow Mapping

Light Source

Camera

a

b
Properties

• **Pro**
  – Very flexible: can handle arbitrary shadow casters
  – Can handle semi-transparent shadow casters
  – Maps well to hardware

• **Con**
  – Aliasing problems occur as shadow map samples are in general not at same positions as camera samples
  – Aliasing is a principal problem of shadow maps – can be reduced but not avoided
  – Difficult question where to place the shadow map

• **Performance**
  – One additional render pass per light source
  – Moderate memory consumption for shadow maps
Dueling Frusta and Aliasing

- **Types of Aliasing**
  - Projective aliasing: light ray almost parallel to surface
  - Perspective aliasing: perspective shortening of view frustrum
Quantifying Sampling Error

\[ d = d_s \frac{r_s \cos \beta}{r_i \cos \alpha} \]
Shadow Map Filtering

- **Percentage-Closer Filtering**
  - Map area representing pixel to texture space
  - Stochastically sample pixel to find percentage of surface in light

![Shadow Map Diagram]

Pixel (in texture space) possibly enlarged
Percentage-Closer Filtering
Properties

• **Pro**
  – Display undersampled data in shadow map in a less disturbing way
  – Fools our perception: Blurring easily mistaken for penumbra in many situations

• **Con**
  – Not geometrically correct soft shadows with penumbra, just blurred, undersampled shadow map
  – Computationally more expensive than normal shadow maps expensive
Parallel Split Shadow Maps (PSSM)

- **Shadow map reparametization**
  - Optimize distribution of shadow map texels

- **Idea of PSSM: Split depth range**
  - Use different shadow map for every layer
Split Plane Positioning

\[ C_i^{\text{uniform}} = n + \frac{(f - n)i}{m} \]

\[ C_i^{\text{log}} = n \left( \frac{f}{n} \right)^i \]

\[ C_i = \frac{C_i^{\text{log}} + C_i^{\text{uniform}}}{2} \]
Properties

• Pro
  – All advantages of shadow maps
  – $n=3$ or $4$ mostly solves issues with perspective shadow map aliasing

• Con
  – Projective aliasing remains a problem

• Performance
  – $n$ additional render pass per light source (but can use multiple render targets)
  – $n$ times memory consumption for shadow maps (but can compensate by reducing resolution due to better usage)
Further Reading

Perspective Shadow Maps:

Volumetric Objects:
HYBRID RENDERING
Hybrid Rendering Techniques

• **Rasterization for primary rendering**
  – Efficiently maps to hardware
  – Resolve visibility using z-buffer

• **Ray-tracing for lighting computation**
  – Ray-scene intersection often too slow for real-time applications
  – Particularly true when sampling large area lights or large opening angles (=many rays per pixel are needed for soft shadows)

• **Core idea**
  – Approximate large number of ray-scene intersections by a small number (often one) cone-scene intersections
  – Intersect against approximate representation of scene that allows fast cone intersection
  – Sufficiently accurate for many lighting purposes
Overview of Techniques

- **Screen Space Ambient Occlusion**
  - Consider z-buffer as a surface that approximates the scene
  - Use ray-tracing in the z-buffer to determine ambient occlusion

- **Sparse Voxel Octree Cone Tracing**
  - Convert entire scene to a view-dependent sparse voxel representation
  - Build mip-map like pyramid over the voxel structure
  - Use “diagonal ray-tracing“ in the voxel pyramid to approximate cone-scene intersection

- **Distance Field Ray Traced Shadows**
  - Build view-dependent voxelized scene representation where voxel contains the distance to the nearest mesh
  - Can be used as acceleration structure for approximate ray-tracing
  - Can be used to calculate ambient occlusion
Ambient Occlusion

- **Calulates shadows against assumed constant ambient illumination**
  - Idea: in most environments, multiple light bounces lead to a very smooth component in the overall illumination
  - For this component, incident light on a point is proportional to the part of the environment (opening angle) visible from the point
  - Describes well contact shadows, dark corners

![Diagram of ambient occlusion with labels: N, α, r, and assumption of constant light outside radius r]
AO Using Ray-Tracing

- Computation using Ray-Tracing straight forward
  - Start at point P
  - Sample N directions \((D_1-D_N)\) from upper hemisphere
  - Shoot shadow rays from \(P\) to \(D_i\) with maximum length \(r\)
  - Count how many rays reach the environment
  - Gives correct result in the limit, but requires many rays to avoid noise (i.e., slow)

![Diagram](image-url)
AO Using Ray-Tracing
Screen Space Ambient Occlusion

- Can we approximate ambient occlusion in real-time?
- Ray-scene intersection too slow

- **Idea: Use z-buffer as scene approximation**
  - Horizontal and vertical position give position of point in x,y-direction (camera space)
  - Z-buffer content gives position of point in z-direction (camera space)
  - Contains discrete representation of all visible geometry
  - Use ray-tracing against this simplified scene
Screen Space Ambient Occlusion

camera

corner

game

geometry outside viewport?

z-buffer

fake corner?
Properties

- **Pro**
  - Allows approximate computation of ambient occlusion in real-time on consumer hardware
  - Replaced „ambient term“ of Phong model in most real-time applications (practically highly relevant)

- **Con**
  - Relies on scene approximation that may generate artifacts
Further Reading


Louis Bavoil, Miguel Sainz and Rouslan Dimitrov, “Image-Space Horizon-Based Ambient Occlusion“, Presentation at SIGGRAPH 2008

Louis Bavoil, Miguel Sainz, “Multi-Layer Dual-Resolution Screen-Space Ambient Occlusion”, Presentation at SIGGRAPH 2009
Recap

• **Today**
  – Re-visited shadows from a rasterization perspective
  – Shadow volumes
  – Shadow maps, parallel split shadow maps
  – Screen space ambient occlusion

• **Next Lecture**
  – Sparse Voxel Octree Cone Tracing
  – Distance Field Ray Traced Shadows