Shadows
Light and shadows

- Shadows increase realism:

![Zaxxon (1982)](image1.png)

![Cry Engine](image2.png)
Light and shadows

- Shadows increase realism
- Shadows help you perceive:
  - hidden objects
Light and shadows

- Shadows increase realism
- Shadows help you perceive:
  - hidden objects
  - the relative position of objects
Light and shadows

- Shadows increase realism
- Shadows help you perceive:
  - hidden objects
  - the relative position of objects
  - the object shape
Light and shadows

- Constraints for real-time shadows
  - Light sources: Dynamic
  - Shadow Casters: Dynamic
  - Shadow Receivers: Dynamic

Doom 3 (2004)
Light and shadows

- 2 kinds of shadows:
  - Hard shadows
    - Point light source
  - Soft shadows
    - Extended light source
Hard shadow

- Point light source
- A point is *in shadow* if it is not visible from the light source
Soft shadow

- 3 areas:
  - Shadow: light source completely hidden
  - Penumbra: light source partially hidden
  - Lit: light source completely visible
Computing hard shadows
Shadows/visibility

- A point is lit if it is visible from the light source

- Computing shadows = visible surface determination
Flat shadows

- Draw the graphics primitives again, projected on the ground
Flat shadows+/-

+ Fast, easy to code
- No self shadows, no shadows on curved surfaces, no shadows on other objects

MIT EECS 6.837, Durand and Cutler
Using textures

- Separate between occluder and receiver
- Draw a picture of the occluder, seen from the light source
- Use it as a texture on the receiver

From the light source

From the viewpoint

Möller & Haines “Real Time Rendering”
Modern shadow algorithms

- Shadow Maps
  - Image space approach
- Shadow Volumes
  - Object space approach
Shadow maps

1. **Offscreen** rendering from the light source
   - Keep z-buffer in a texture

2. Rendering from the view point
   - Transform current pixel into light space coord.
   - Compare current depth with depth in texture
   - Change lighting depending on visibility test
Shadow maps: step 1

- offscreen rendering from the light source:
  - Transformation + projection matrix
  - Light space coordinates
  - Store depth into an FBO
- FBO -> texture
Standard rendering

Vertex shader:
- Compute projection in screen space
- And in light space

Fragment shader:
- Interpolate coordinates
- Coord. texture shadow map
- $z =$ distance light source
- $z$ from shadow map
- Comparison
- $\Delta$ Coord. texture $= [0,1]^2$
Shadow maps: comparison

- $z_{\text{shadowMap}} < z_{\text{computed}}$
  - In shadow
  - Ambient lighting only
- $z_{\text{shadowMap}} = z_{\text{computed}}$
  - Lit
  - Ambient + Diffuse + Specular
- $z_{\text{shadowMap}} > z_{\text{computed}}$
  - Should not happen, in theory
Shadow maps: 1st picture
“it’s not a bug, it’s a feature”

What’s happening?
- Comparison $z$ stored/interpolated $z$
- $z$ value constant for each pixel
- Self-shadowing

Solutions:
- Comparison with $z+\varepsilon$ (bias)
- Draw only back-sided surfaces
Back–sided surfaces only

- Easy: `glCullFace(GL_FRONT);`
Back-sided surfaces only

A few issues with self-shadowing (in reverse)
Still need bias for comparisons.
Back-sided surfaces only
Behind the Scenes
Small constant bias (5e−3)
Medium constant bias ($1.5e^{-2}$)

Shadow discontinuities
Other bias methods

- Slope–dependent: \( \tan(\text{angle N,L}) \cdot a + b \)
  - \( b > 0, \ a = ? \)
- Relative: \( z1 \cdot (1 - \epsilon) < z2 \)
Projection / light source

- It’s a *projection*:
  - Must divide by $w$
- What does it mean if $w < 0$?
  - What should we do?
- What should we do if we’re outside shadow map?
  - How can we check?
Shadow maps: pro & cons

- **Pros**
  - Easy to implement
  - Works, regardless of the geometry of the scene
  - Cost does not depend on scene complexity

- **Cons**
  - Several (≥ 2) scene rendering
  - Omni-directional light sources?
  - Sampling/aliasing
    - Increasing shadow map resolution is not enough (light source facing viewer)
Aliasing issues: solutions

- Increase shadow map resolution
- Focus shadow map on visible parts of the scene
- Adapt sampling (warping)
  - Depending on light-source distance
- Multi-resolution Shadow maps
  - Cascading shadow map
Focus the shadow map

Increases the practical resolution
**Warping** for shadow maps

- **How?**
  - **Linear projection**
    - Not centered on the light source
    - Optimized based on view frustum + LS position
  - TSM, LiSPSM...

Uniform sampling in z

Variable sampling in z
Cascading shadow maps
Cascading shadow maps
Cascading shadow maps
Cascading + warping
Shadow Volumes algorithm

1. For each *shadow casters*, build a **shadow volume**

2. For each fragment, **count** how many times we enter/leave a shadow volume
   - > 0 : in shadow
   - = 0 : lit
Shadow Volumes algorithm

- Building a shadow volume
  - Silhouette of each object from the light source
  - Infinite quads touching
    - the light source
    - Each silhouette edge

- Counting entering/leaving
  - Use the *stencil buffer*
  - Use the orientation of each shadow quad for the sign
Extract the silhouette?

- Silhouette of each object from the light source

How? 1mn
Building semi-infinite quads?

How? 1mn
How do we count?

- Use the Stencil buffer
  - Shadow volume side visible, front-facing: +1
  - Shadow volume side visible, back-facing: −1

- 2 rendering pass:
  - First front-facing, then back-facing
  - `glCullFace(...)`

- 1 rendering pass:
  
  ```c
  glStencilOpSeparate(GL_FRONT, GL_KEEP, GL_INCR_WRAP, GL_KEEP);
  glStencilOpSeparate(GL_BACK, GL_KEEP, GL_DECR_WRAP, GL_KEEP);
  ```
Z-pass by example: how the stencil buffer is used
Z-pass: issue

- What if the eye is in shadow?
Z–fail

- Have a lit point as reference
- A point at infinity must be lit
- Need to cap the shadow volume

Simply invert z-test and invert stencil inc/dec

Near capping

Far capping
Z-fail by example
Shadow volumes: pro&cons

- **Pros:**
  - Sharp shadows
  - Arbitrary positions for light source/camera
  - Robust (if well programmed)

- **Cons:**
  - Silhouette computation (CPU/GPU)
  - Requirements on scene geometry (manifold, closed surfaces)
  - Rendering the scene twice, + the shadow volumes
Overdraw

Shadow volumes

CC Shadow volumes
Soft shadow computations
Soft shadows

- More complex
  - **Point-to-area** visibility, with continuous value
    - Instead of binary point-point visibility
    - silhouette?
Soft shadows

- More complex
  - **Point-to-area** visibility, with continuous value
    - Instead of binary point-point visibility
    - silhouette?
  - Shadow of the sum ≠ sum of shadows
    - A hides 50% and B hides 50%, A+B doesn’t hide 100%
Soft shadows

- More complex
  - **Point-to-area** visibility, with continuous value
    - Instead of binary point-point visibility
    - silhouette?
  - Shadow of the sum ≠ sum of shadows
    - A hides 50% and B hides 50%, A+B doesn’t hide 100%

- Many algorithms
  - With varying accuracy
    - Approximating the *shadow casters*
    - Precomputations (*Precomputed Radiance Transfert*)

- With varying speed
Soft shadows through sampling

- Accumulating shadows:
  - Compute several hard shadows
  - Add them, average the results
  - *accumulation buffer*
  - Needs many samples
    - Computation time proportional to # échantillons

4 échantillons

1024 échantillons
For each silhouette edge:
  • Compute volume around penumbra (wedge)
  • For each pixel in this wedge
    • Compute attenuation coefficient

Beautiful, realistic, expensive

Penumbra wedges [Sig03] U. Assarson, T. Möller
Object/image methods

Rendering Fake Soft Shadows with Smoothies [SoR03]
E. Chan, F. Durand
Shadow mapping extension

- **Percentage Close Filtering (PCF)**
  - Filter shadow map around sampling point
  - Possible GPU speed-ups (2x2 kernel)
  - Pre-filtered, stored in mip-map
**Shadow mapping extension**

- *Percentage Closer Soft Shadows (PCSS)* [Fernando 05]
  - Compute kernel size first, by sampling shadow map
  - Filter using PCF (or extensions)
1. Blocker search
2. penumbra size

\[ w_{\text{penumbra}} = \frac{p_z^s - z_{\text{avg}}}{z_{\text{avg}}} w_{\text{light}} \]
PCSS

3. filtering

Filter region
(size $\sim w_{\text{penumbra}}$)

(here, occlusion = 50 %)
PCSS: issues (1)

- 1. blocker search
- 2. penumbra size
- 3. filtering

2 steps requiring several access to shadow map
PCSS : issues (2)

- Easy, quite fast
- Visually pleasing results
  - For a small light source
- No physical realism
- Visible artefacts
  - For large penumbra width
  - If occluders hidden from center of light source
  - For non-flat occluders
Shadow mapping extensions

- Percentage Closer Soft Shadows (PCSS)
  [Fernando 05]


PCSS

PCF
Soft shadow maps

Soft Shadow Maps [AHLHHS06] Atty et al.

(a) Scene view
(b) Discretizing occluders
(c) Soft shadows from one micro-patch
(d) Summing the soft shadows
Back-projection

- Shadow map = object discretization
- Compute shadow of discretized object
- Realistic, real-time, animated scene
- [Atty06] et [Guennebaud06]
Back-projection
Ambient occlusion
Motivation

- Approximating the occlusion under distant lighting
  - Ambient term taking visibility into account
- Perceptually related to depth, curvature and spatial proximity
Definition

- Integral of visibility over hemisphere $\Omega$:
  \[
  A_p(\vec{n}) = \frac{1}{\pi} \int_{\Omega} V_p(\vec{\omega})(\vec{n} \cdot \vec{\omega}) d\omega
  \]
  - Cosine term $\Rightarrow$ diffuse lighting
  - Usually, attenuation depending on the distance to P
Computing the integral

- Sampling
  - Precomputation (*ray-casting*)
  - Store in a texture

+ Rendering at no extra cost
- Slow precomputation
- Static scene

GPU Gems, chap 17
Computing the integral

- **Screen-Space Ambiant Occlusion (SSAO)**
  - Use the *depth buffer* as an approximation of the scene
  - For each pixel, sample the hemisphere on the GPU
  - Filtering for noise reduction

+ Independent form scene complexity
+ No pre-computation
+ Dynamic scene
- Longer rendering

Cry Engine 2
Deferred shading

- Fragment shaders get expensive:
  - Complex materials, textures, indirect lighting...

- Pb. for complex scenes/multi-layers:
  - Shading for all surfaces
  - Even if they’re invisible
  - Z-buffer test after the fragment shader

- Need: visibility before shading materials
  - Theoretically impossible
  - Solution: deferred shading
Deferred shading

- 1\textsuperscript{st} pass: rendering into 3–4 aux. buffers

- 2\textsuperscript{nd} pass: compute shading using these buffers
Deferred shading + SSAO

- SSAO:
  - Needs a geometry buffer
  - Is expensive: must reduce number of calls

- Deferred shading:
  - Has a geometry buffer
  - Did visibility as pre-computation

- Good match of algorithms
Beyond SSAO

Approximating Dynamic Global Illumination in Image Space
Ritschel et al. 2009