Real Time Shadows in 3D Computer Graphics

Stefan Seipel
Pictures adapted from Hans E. Molin

Reasons to use shadows?

- Shadows increase the level of reality
  - The real world has shadows
  - Shadows provide important depth information
- Shadows enhance spatial perception
Shadows in computer graphics

- Do we have to calculate a shadow?
  Don’t you get shadows for free?
- In ray-tracing you do
- With the method used in real-time 3D computer graphics you don’t

Common real-time shadow methods

- Analytical methods
  - Projected Planar Shadows
  - Shadow Caster Projections
  - Shadow Volumes
- Image/Pixel-based methods
  - Shadow Mapping
Projected Planar Shadows / Ground-Plane Shadows

- Fastest method
- Easiest method
- Casts shadows only on one flat ground-plane
- Very unrealistic

Projected Planar Shadows / Ground-Plane Shadows

- Render a ground-plane
- Render an object
- Then render the object again, but this time
  - Projected onto the plane
  - Without light, so that the shadow is black
  - Half transparent (using blending), to avoid completely dark shadows
  - Avoid multiple “darkening” on one spot by using ordinary z-buffer checks
Requirements of good, ”real” shadow methods

- Objects cast shadows on other objects
- Self-shadowing
- Multiple light sources
- Fast enough for real time rendering
- Can be optimized with hardware
- (Penumbra shadows)

Shadow Volumes

- Shading object (Occluder, Shadow caster)
- Surface outside of shadow volume (lit)
- Shadow volume
- Partially lit object (Receiver)
- Surface inside of shadow volume (in shadow)
- Light source
- Position of eye
Shadow Volumes

- Generate shadow volumes
  - Add new polygons that represent the edges/surfaces of the shadow volume
    - Always works
    - Computationally heavy

or

- Move the objects back-facing vertices away from the light source
  - Can be done with a vertex shader
  - Only works if the object is well tessellated
Shadow Volumes

Render shadow volumes using stencil buffer or a texture
- The stencil buffer is a buffer that can be used to store data about pixels in the rendered scene. It can be used for example to restrict rendering of certain pixels
- Start with setting all values in the stencil buffer to the number of shadow volumes the eye is in (i.e. 0 if eye is not in a shadow)
Shadow Volumes

- The shadow volumes are drawn in two steps to handle overlapping shadow volumes
  - Render front-facing polygons
    - For every pixel in the frame buffer
      - increase the corresponding stencil buffer value by one for every polygon that the imagined line between the eye and the pixel passes through (enters a shadow volume)
  - Render back-facing polygons
    - For every pixel in the frame buffer
      - decrease the corresponding stencil buffer value by one for every polygon that the imagined line between the eye and the pixel passes through (leaves a shadow volume)

We can now render the scene using the stencil buffer as a light occlusion mask

- The stencil buffer indicates for every pixel if it’s lit or in a shadow
  - If stencil value = 0 then the pixel is lit (outside of any shadow volume). Render with light
  - If stencil value != 0 then the pixel is in a shadow (inside of one or more shadow volumes). Render without light
Shadow Mapping

- Render the scene from the lights point of view and save the depth values to a texture
  - The result is a “shadow map”
  - Every pixel in the map indicates its distance from the light source

Shadow Mapping

- Render the scene from the eye’s point of view
  - For every pixel in the rasterization
    - Decide the pixels XYZ location relative to the light source
    - The location is calculated using the projection matrix used when rendering the shadow map
    - Compare the depth value on position XY in the shadow map (closest surface to light source) with the pixels distance to the light source
Shadow Mapping

- Shadow map comparison
  - Let $A = Z$-value (from light source) on position $XY$ in the shadow map
  - Let $B = Z$-value (from light source) for the current pixel to render
  - If $B > A$ we know that something is closer to the light source and hence the pixel is in shadow
  - If $B \approx A$ the pixel is lit (approximately equal because of rounding error)

Soft Shadows

- Area-light source
- Point-light source
- Umbra
- Penumbra
- Occluder
Soft Shadows

- Soft Shadows can be accomplished with
  - Shadow volumes by having multiple light sources that each produce a shadow volume for every object
  - Shadow mapping using multiple light sources that each produce a shadow map

Volumes vs. Mapping

- Advantages of shadow mapping
  - Faster than shadow volumes (most of the time)
  - You don’t have to add extra vertices, calculate extra polygons or calculate an object’s silhouette. You hardly need to know anything about the objects to be able to use shadow mapping
  - More optimization options: different resolutions, depth precision, filtering etc.
Volumes vs. Mapping

- Advantages of shadow volumes
  - Doesn’t suffer from aliasing effects and rounding errors as shadow mapping does
  - Can achieve omni-directional shadow casting, shadow mapping requires at least 6 shadow maps to do this

Program Examples

- Shadow Volumes
- Soft Shadows using multiple Shadow Volumes
- Shadow Mapping
More information

- Shadow Mapping
- Shadow Volumes
- Planar shadows
  http://www.opengl.org/developers/code/features/StencilTalk/index.htm