Real Time Shadows in 3D Computer Graphics

Hans E. Molin

Reasons to use shadows?
- Shadows increase the level of reality
  - The real world has shadows
  - Shadows help to determine the relative position of objects, particularly depth order and the height of objects above the ground plane.
- Effects like in the movie Dracula, where the shadow acts independently from the shadow caster

Depth Information with shadows

Shadow effects

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 known as synthetic-camera model you don't!

Common real-time shadow methods
- Analytical methods
  - Projected Planar Shadows
  - Area subdivision
  - Shadow Volumes
- Image/Pixel-based methods
  - Depth-Incorrect Soft (Planar) Shadows
  - Object ID/Priority Buffers
  - 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 “true” shadow methods**

- Objects cast shadows on other objects
- Objects cast shadow on themselves
- Multiple light sources
- Fast enough for real time rendering
- Can be optimized with hardware

**Shadow Volumes Overview**

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

**Adding Shadow Volumes**

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

**Extrude Shadow Volumes**

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

- Shading object (Occluder, Shadow caster)
- Position of eye

**Shadow Volumes using stencil buffer**

- 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).

**Rendering the 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).

**Shadow Volumes: Use the stencil buffer**

- 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 light's point of view and save the depth values to a texture:
  - The result is a depth map or "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 pixel's 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 with the pixel's distance from the light source

**Shadow Mapping: Applying the shadows**

- The shadows are rendered in the scene by applying a texture to all objects.
- If it is shadowed, the value of the shadow texture is 0, otherwise it is 1

**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

**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

**Mapping and Volumes Examples**

- Shadow Volumes
- Shadow Mapping

**Performance comparison**

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Shadow Volumes</th>
<th>Shadow Mapping</th>
<th>No Shadows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1280x1024</td>
<td>59</td>
<td>93</td>
<td>97</td>
</tr>
<tr>
<td>1024x768</td>
<td>76</td>
<td>97</td>
<td>98</td>
</tr>
<tr>
<td>512x384</td>
<td>93</td>
<td>97</td>
<td>98</td>
</tr>
<tr>
<td>256x192</td>
<td>97</td>
<td>98</td>
<td>100</td>
</tr>
</tbody>
</table>

**Activating shadows in VRT**

- VRT_ShadowSwitchOn()
- VRT_ShadowSetLookAt(pos, look, up)
- VRT_ShadowSetFOV(fov, near, far)

**Example VRT program with shadows**