- Shadows
- Terminology
  - Occluders: cast shadows
  - Receivers: have shadows on them
  - Point light sources: make hard shadows
  - Area light sources: make soft shadows
  - Umbra: the fully shadowed region
  - Penumbra: the partially shadowed region
- Problem
  - Existing APIs do not completely solve the rendering equation – and do not include shadows
- Solutions
  - Projection shadows
    - Idea: just project the occluder onto a receiving plane
    - How
      - One can derive the needed projection matrix
      - Then apply the projection matrix to the occlude geometry
    - Caveats
      - How to put the result on the receiving plane without being inside it (depth aliasing)?
        - Best solution: render receiving normally and first, then render the projected occlude with z-buffering off, then render the rest of the scene normally
      - What happens when the projected shadow falls outside the bounds of the receiver?
        - Solution: turn on the stencil buffer when rendering the receiver, use it when rendering the occluder
          - Note that this results in part of the shadow “going missing”
      - What if the shadows aren’t opaque?
        - If the object is convex, no problem: render the occluder using transparency.
        - Use backface culling if two polygons are too many.
        - If not, then will have varying numbers of polygons at each pixel
          - Best solution: use the stencil buffer, allow only the first pixel of the occluder to be rendered
      - Why render the shadow every time? Wasteful if shadow don’t move
        - Render the shadow into an offscreen texture that is rerendered only when the shadow (light, occluder, receiver) changes
      - Make sure that the occluder is between the light and the receiver, else errors result
        - Can use the receiver to clip/cull potential occluders
      - What about receivers that aren’t planes?
  - Shadow mapping (Williams 78)
    - Idea
      - Consider the scene from the view of the light source
      - What the light sees is not in shadow
    - How
      - Render the scene from the view of the light source
      - This creates depth values that can be located in 3D: the shadow map
      - When rendering the scene from the eye’s viewpoint
      - Transform depths into the shadow map space
      - If depths are greater than those in the shadow map, corresponding object is in shadow
      - If in shadow, don’t include light from this source
    - Caveats
      - Have to render twice: once for light view, once for eye view
        - Can calculate depth only during light pass
        - Can reuse shadow map if light, occluder, receiver do not move (viewer can move)
      - Precision depends on XY and Z resolution
        - Image aliasing
          - Problem
            - Resolution in light view is not resolution in eye view
            - E.g. when a receiver is normal to one view, and orthogonal to other

- E.g. fragments in eye will not generally match exactly to fragments in light view
- Solutions
  - Can filter neighbors to get average (but this has its limits)
- Depth aliasing
  - Problem
    - Can be hard to tell if an eye fragment is in front of a light fragment, especially if the views are not similar
  - Solution
    - Can add a “bias” constant to solve (works mostly, but not always)
      - “Peter panning”
    - Can change algorithm to use object IDs
      - But then objects will never shadow themselves
- Flicker between frames
  - Caused by changes in view sampling
  - Can regenerate shadow map each frame, and use consistent sampling scheme
- Advantages
  - Linear (2 passes)
  - Works for arbitrary receivers

Shadow volumes

- Idea
  - If the objects are polygonal, then so are the sides of the shadows
  - We figure out what these sides are
  - Then figure out if a fragment is inside the resulting volume
- How
  - For each light
    - For each edge in each polygon
      - Define resulting shadow quadrilateral by
      - Drawing ray from light to each vertex
    - For each ray from the eye (for each eye fragment)
      - Count the number of front facing sides it crosses f
      - Count the number of backfacing sides it crosses b
      - If f-b is positive, then fragment is in shadow
  - Details
    - For each light in each frame
      - Clear stencil buffer
      - Draw scene (without shadow volumes) using only ambient and emission, with z-buffering on
      - Turn off frame and z-buffer updates (leave ztest on)
      - Draw front facing shadow polygons
        - Increment stencil for each fragment
      - Draw backfacing shadow polygons
        - Decrement stencil for each fragment
      - Draw scene (without shadow volumes) using only diffuse and specular
        - Apply diffuse and specular only if stencil value is zero
- Caveats
  - Shadow pierces the front or back clipping planes
- Disadvantages
  - Translucent objects
    - As receiver: we only story the state of one object per pixel
    - As occluder: we know only if we are in our out of shadow
  - Slow
    - Lots of fill processing in all the shadow polygons!