REAL-TIME SHADOW RENDERING

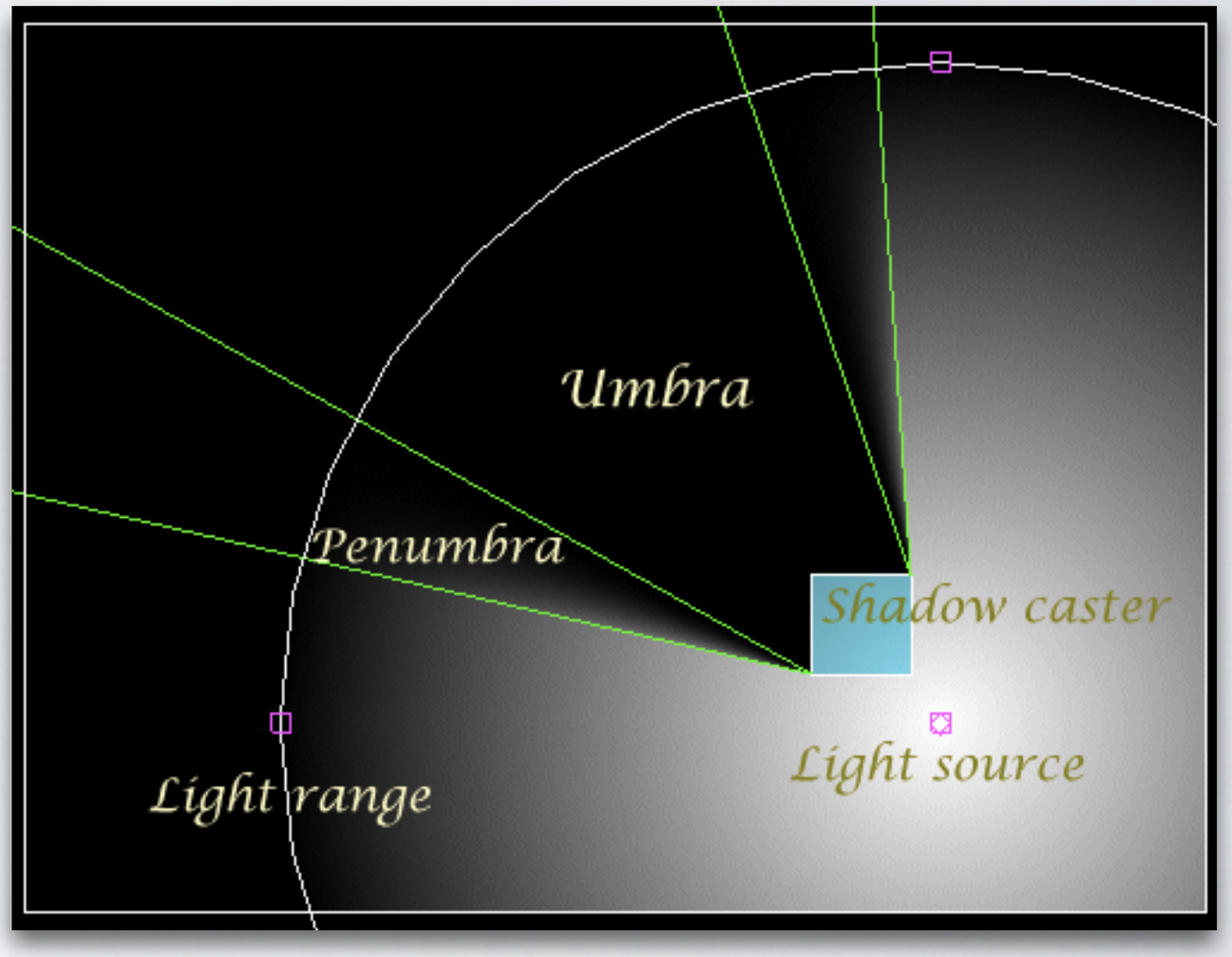
Ju Heqi (Autodesk Inc.)

CONTENT

- Basic Shadow Techniques
- Shadow Map anti-aliasing
- Soft Shadows

DEFINITION

A **Shadow** is an area where the light, coming from the light source, is completely or partially obstructed by an object



Importance of shadows

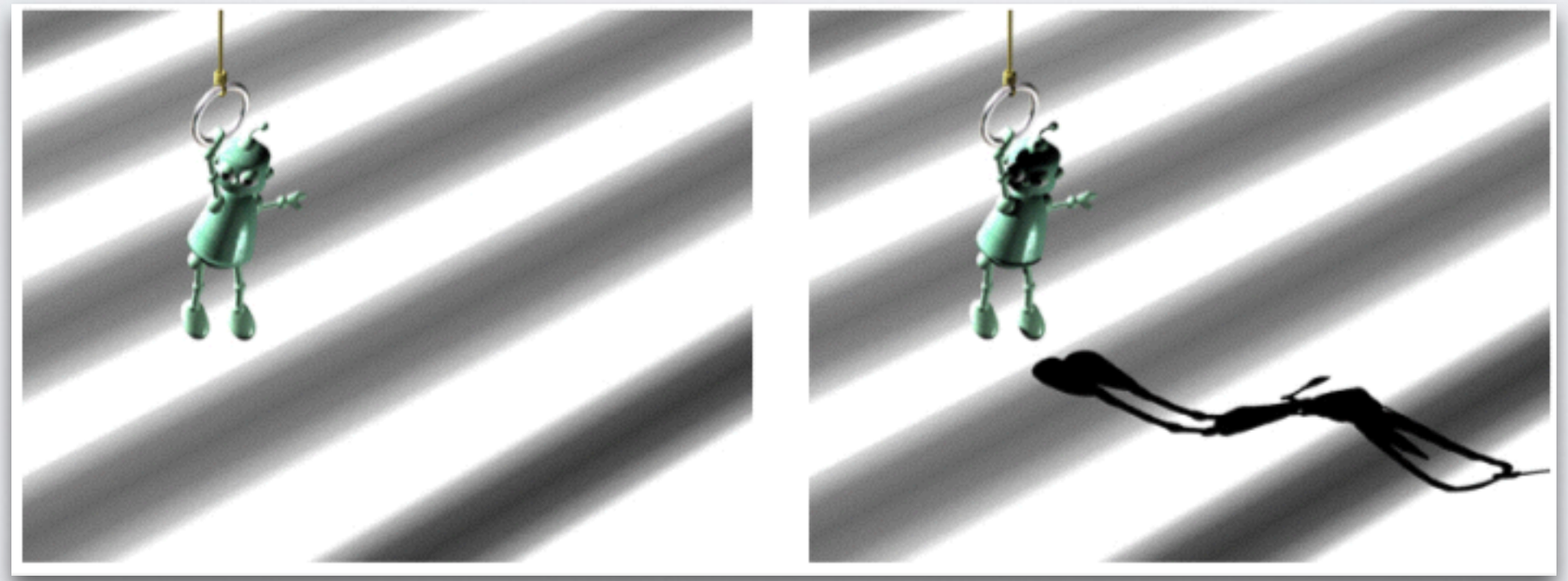
- Shadow helps to improve how realistic an image is;

Importance of shadows

- Shadow provides clues of spatial relationship of objects in the scene;

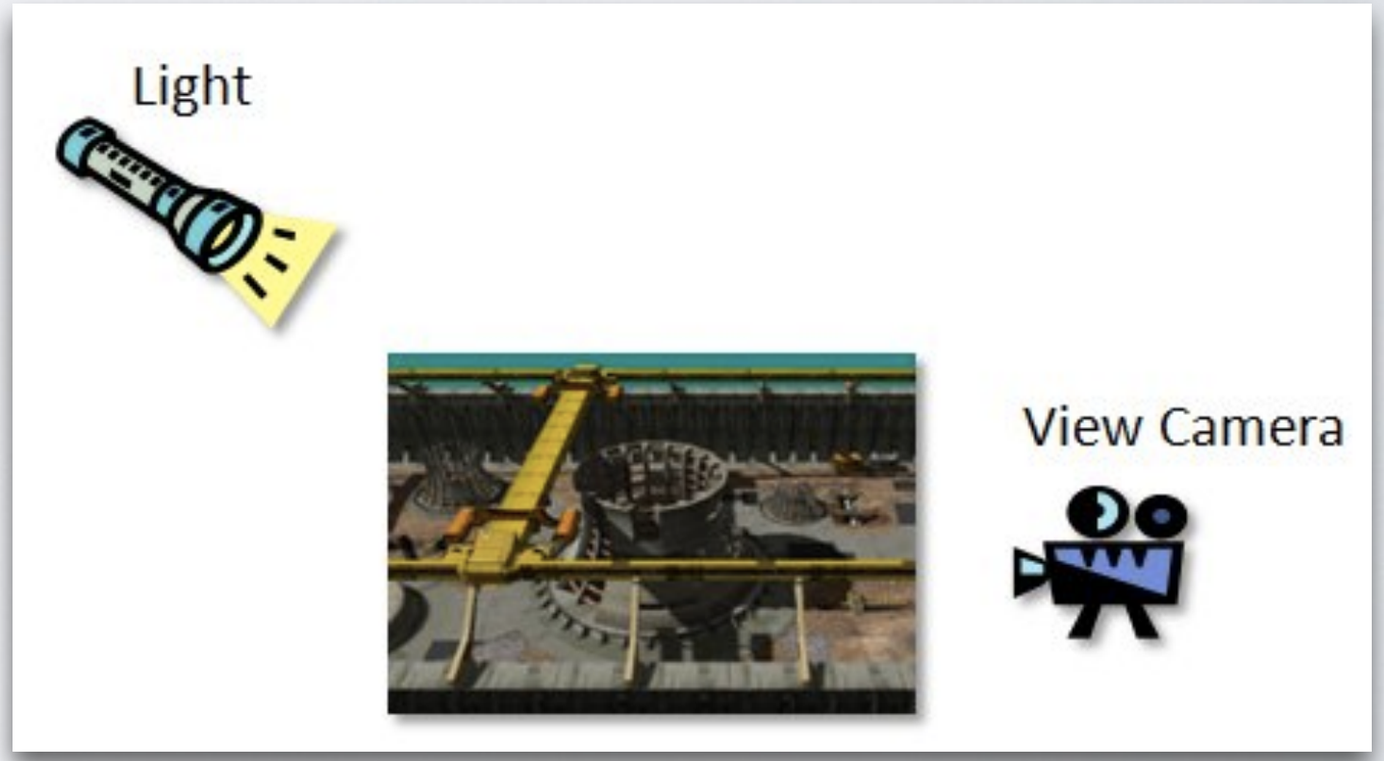
Importance of shadows

- Shadows are also used to identify surface characteristics and details of the the light source.



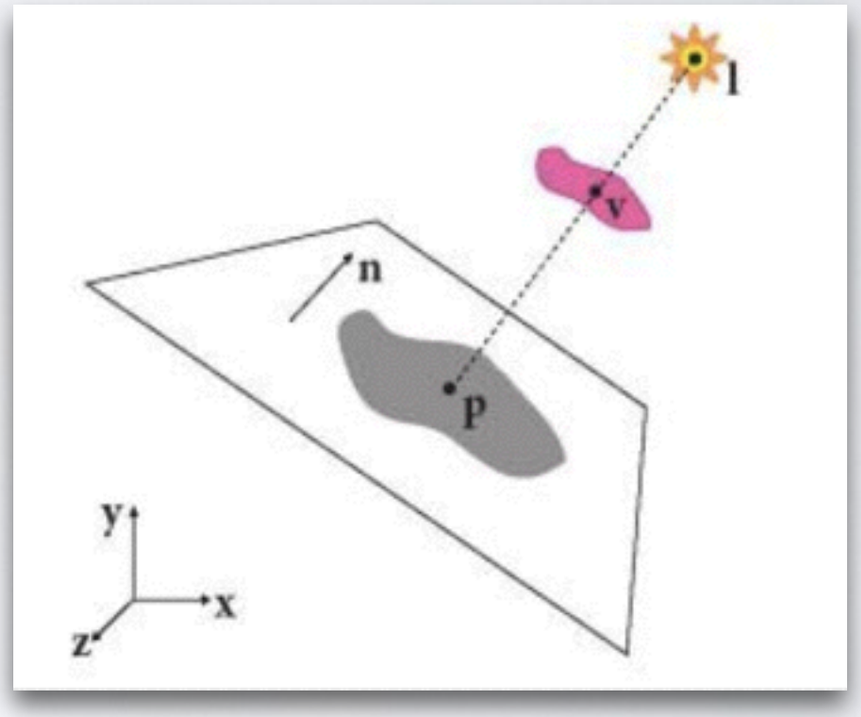
BASIC SHADOWS

- Projection shadow
- Shadow mapping
- Shadow volume



Projection shadow

- an old technique;
- projecting the object's shadow upon the plane/ surface;

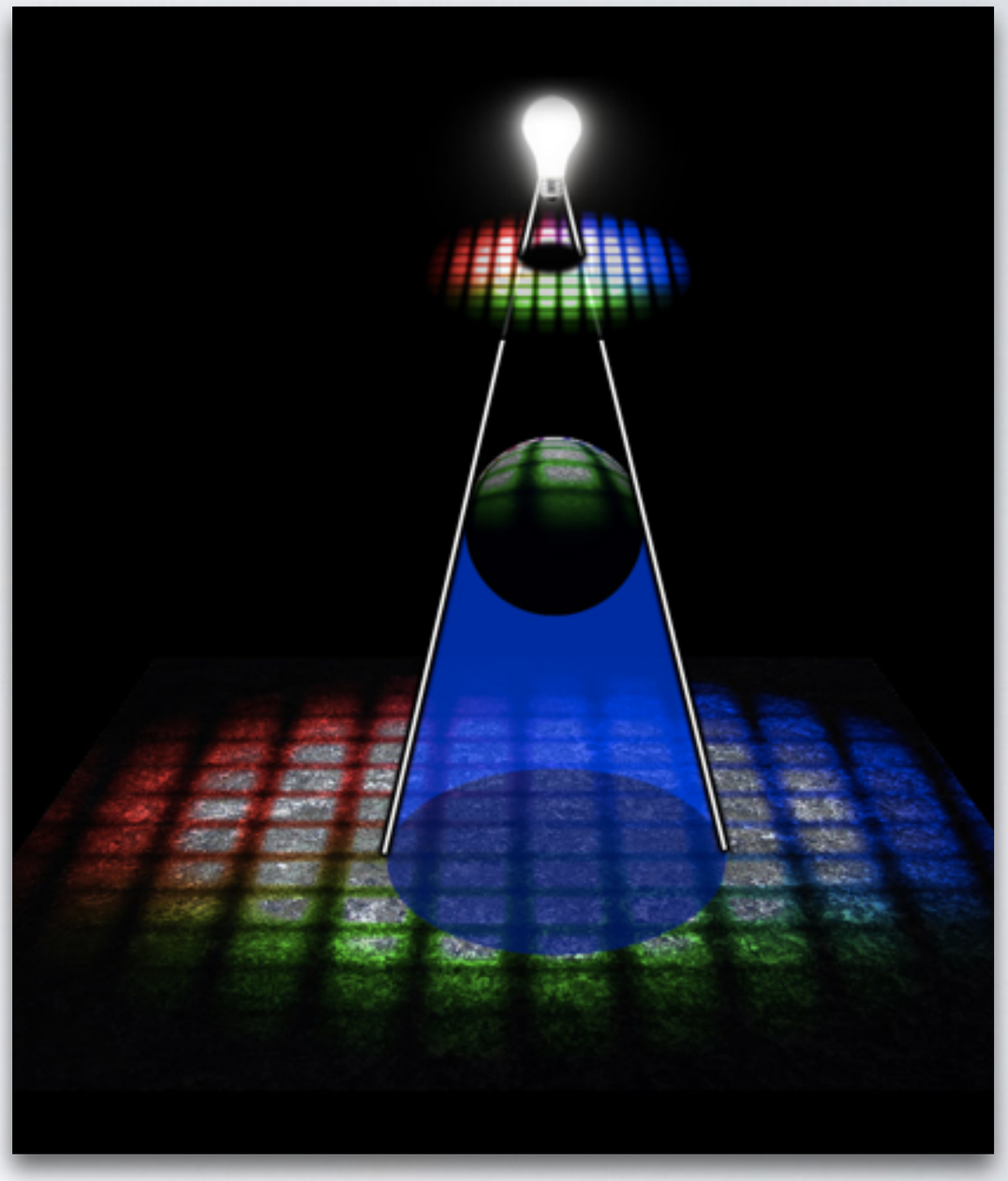


The Projection Matrix

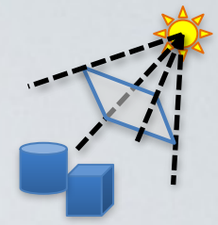
$$\begin{matrix} n \cdot l + d - n_x l_x & -n_y l_x & -n_z l_x & -d l_x \\ -n_x l_y & n \cdot l + d - n_x l_x & -n_z l_y & -d l_y \\ -n_x l_z & -n_y l_z & n \cdot l + d - n_z l_z & -d l_z \\ -n_x & -n_y & -n_z & n \cdot l \end{matrix}$$

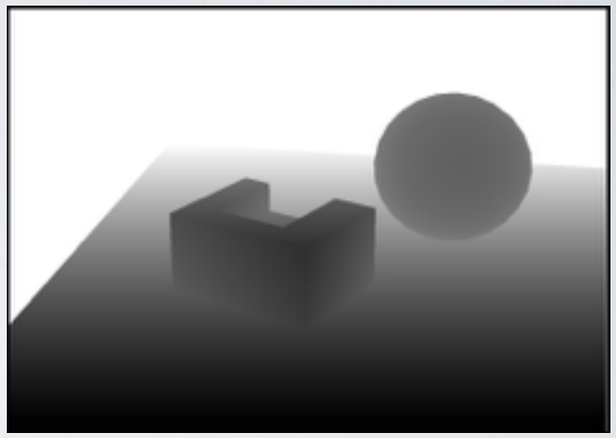
Shadow Texture

- a form of projective texture mapping;
- projecting uv coordinates;
- works on curved surfaces.



Shadow Maps



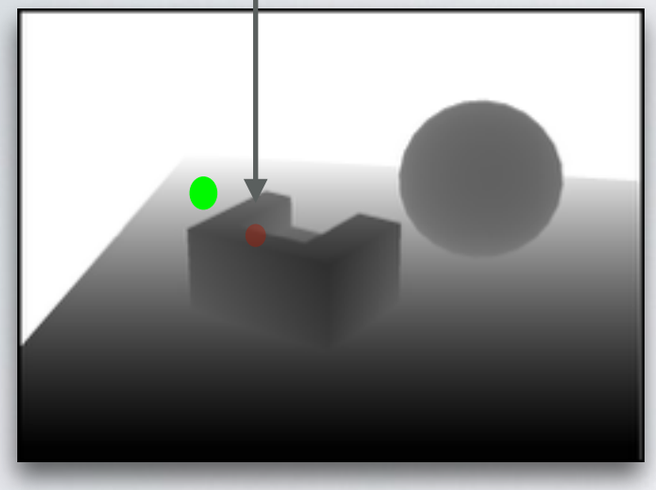


Shadow Map (light's view)

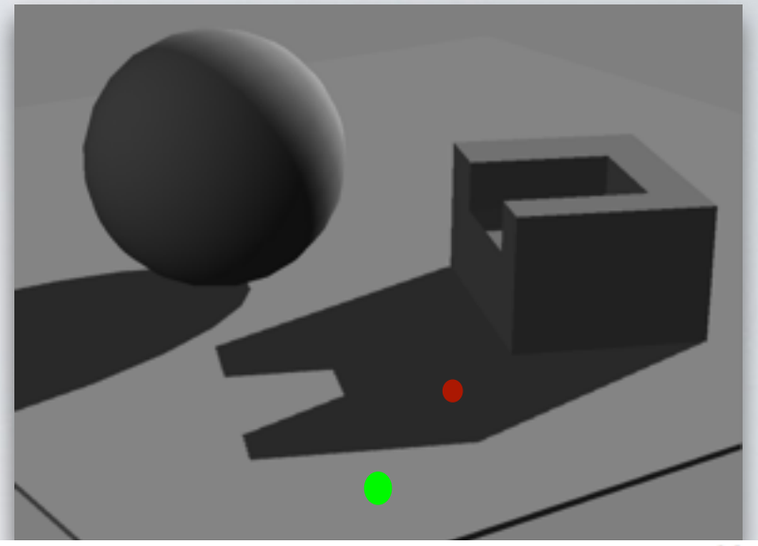
- Rendering image from light source
- Rendering from camera
- Test if the rendered vertex is visible in light's view;
  - if True, vertex is in light
  - else , vertex is in shadow

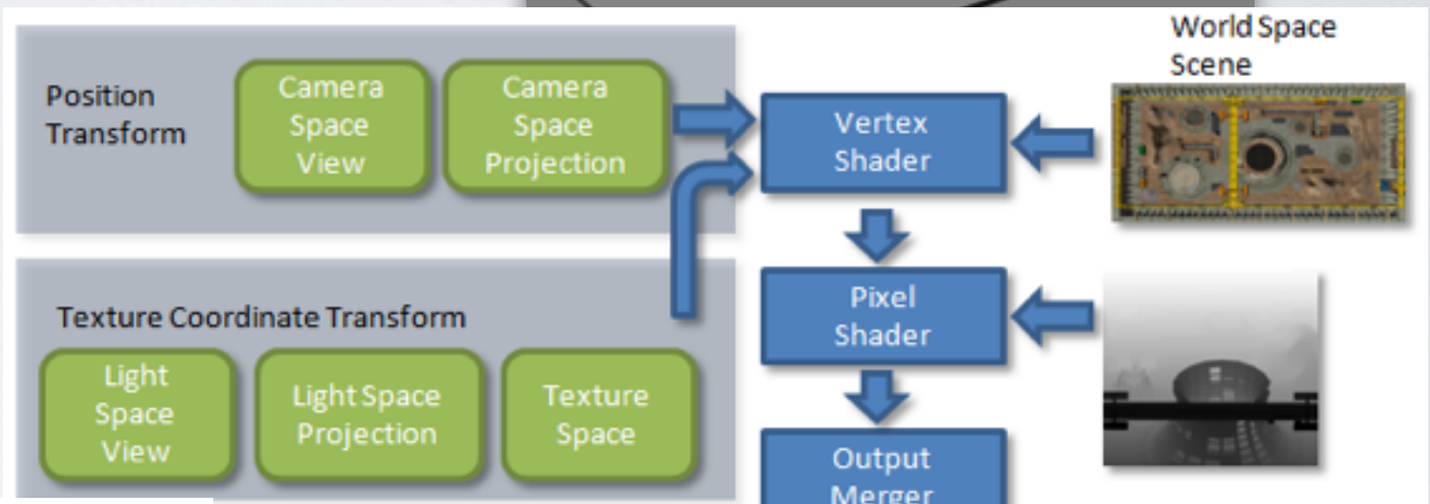
Light's view (shadow map)

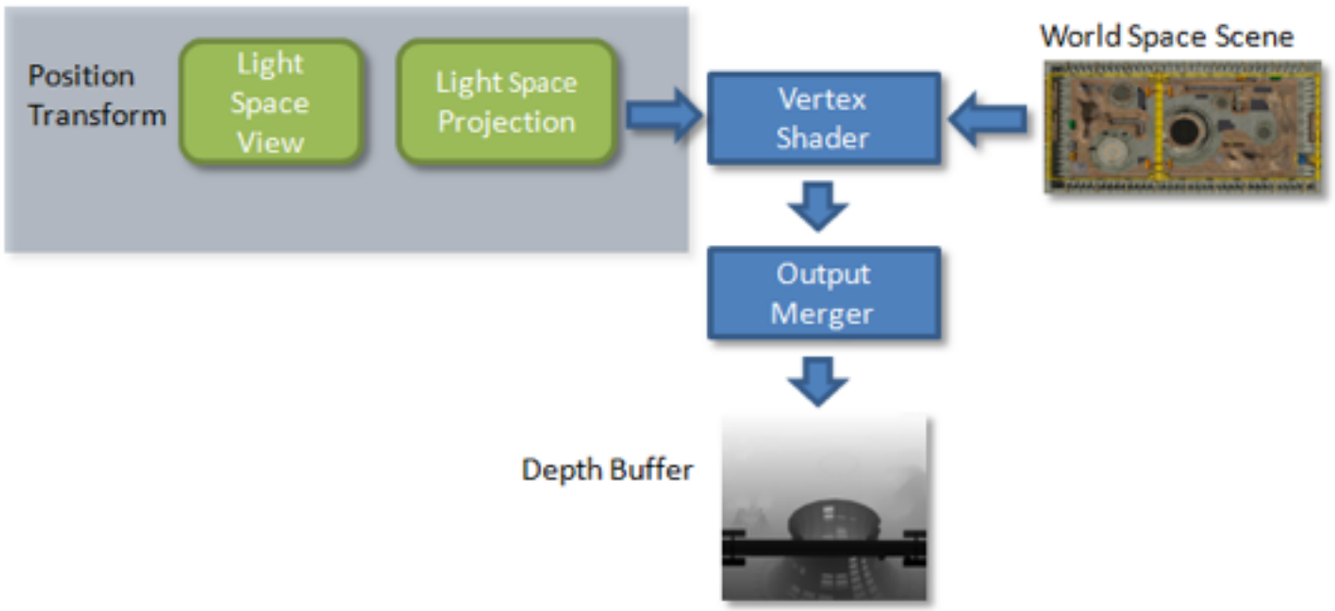
Vertex not represented in shadow map



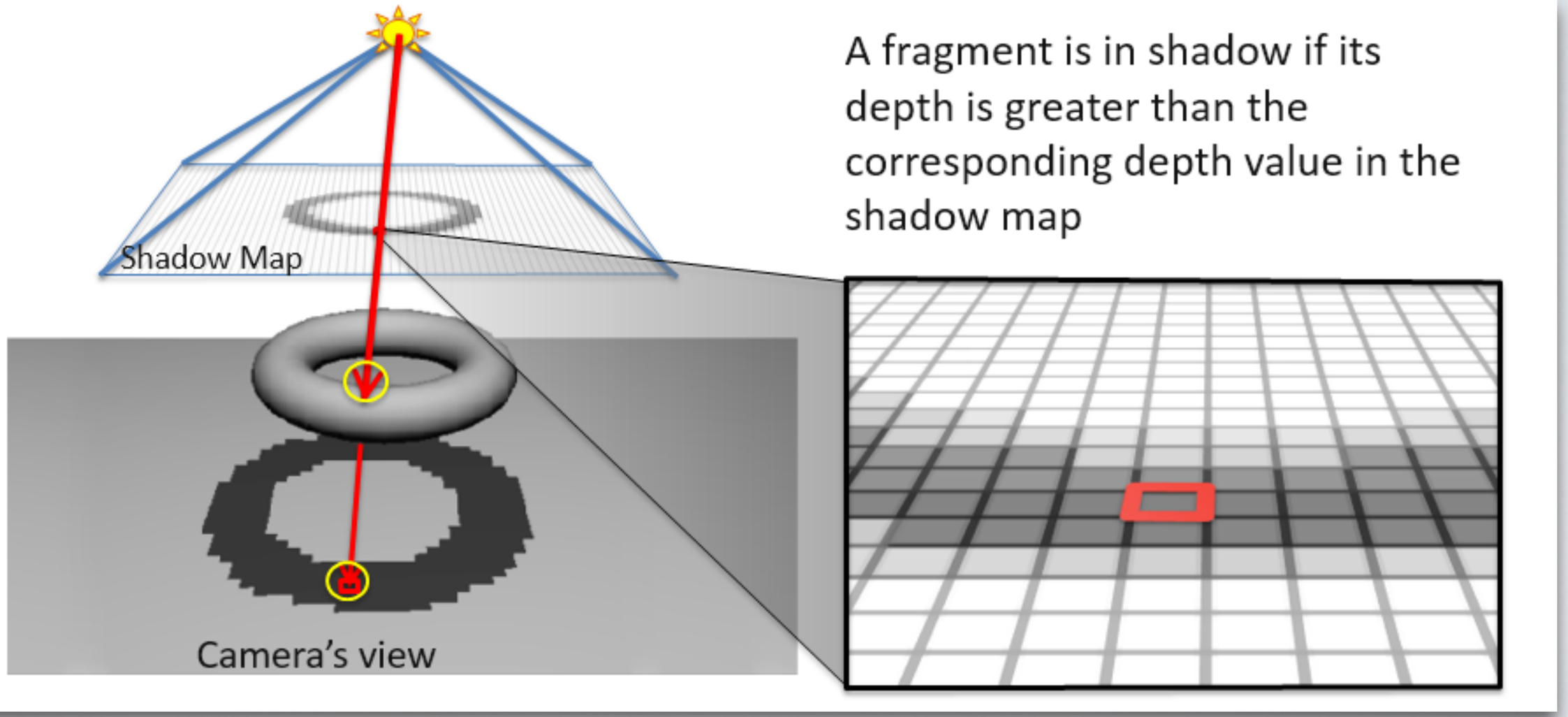
Camera's view







Depth Comparison



Shadow Maps

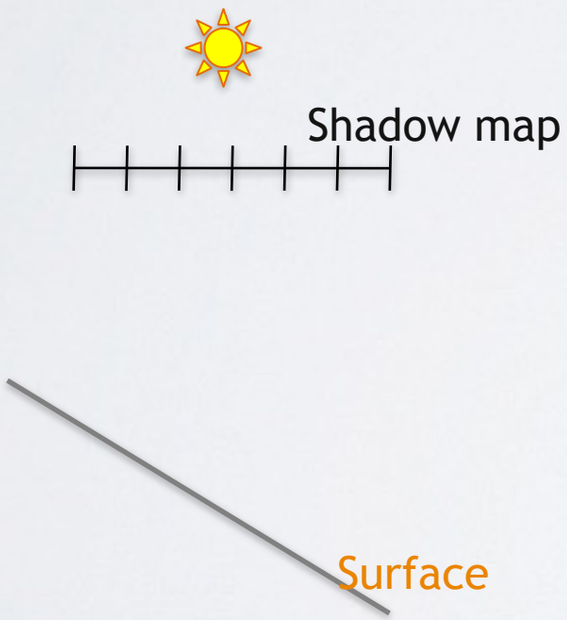
- Pros
  - Very efficient
- Cons ...

Shadow Maps - Problems

- Low Shadow Map resolution results in jagged shadows

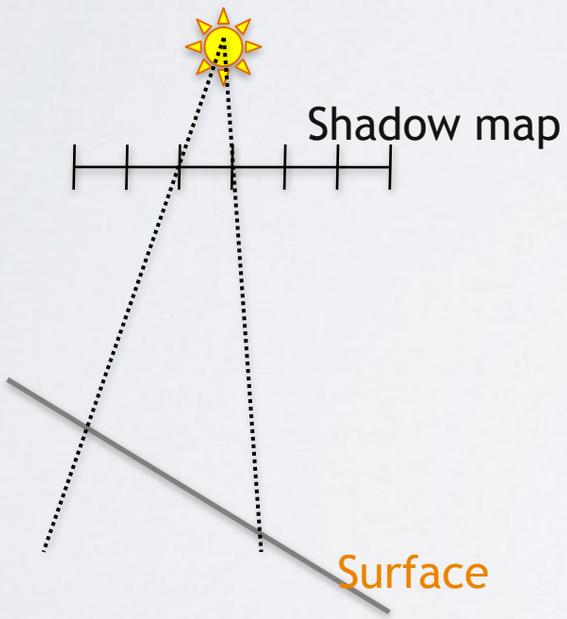
Shadow Maps - Problems

- In addition: A tolerance threshold(bias) needs to be tuned for each scene for the depth comparison



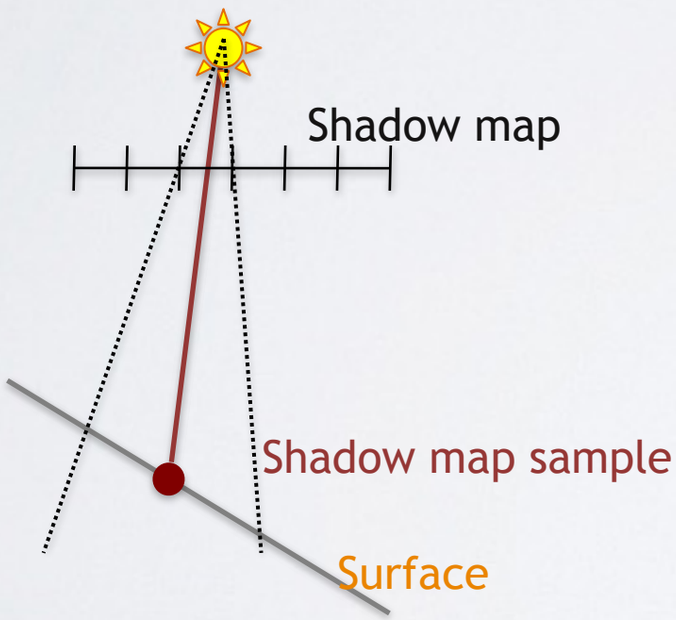
Shadow Maps - Problems

- In addition: A tolerance threshold(bias) needs to be tuned for each scene for the depth comparison



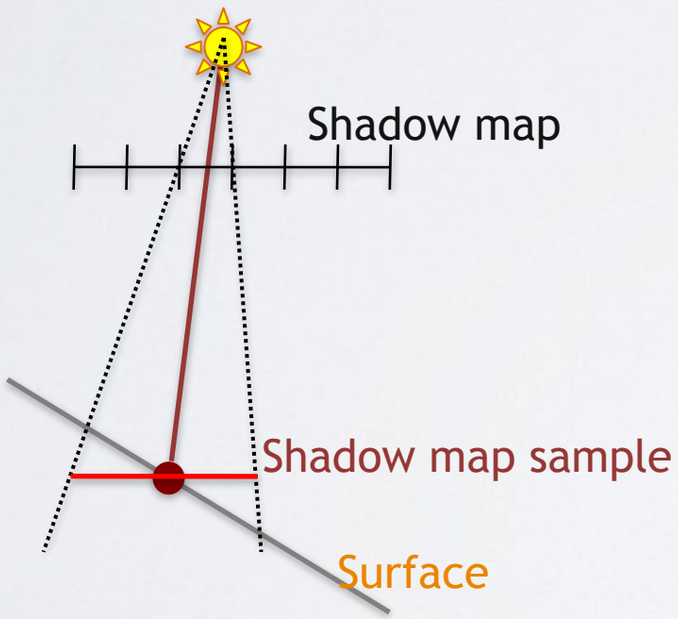
Shadow Maps - Problems

- In addition: A tolerance threshold(bias) needs to be tuned for each scene for the depth comparison



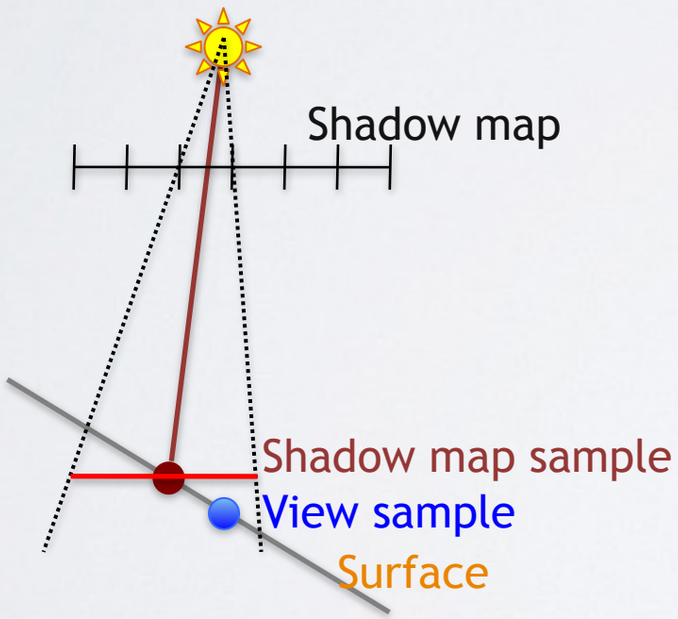
Shadow Maps - Problems

- In addition: A tolerance threshold(bias) needs to be tuned for each scene for the depth comparison



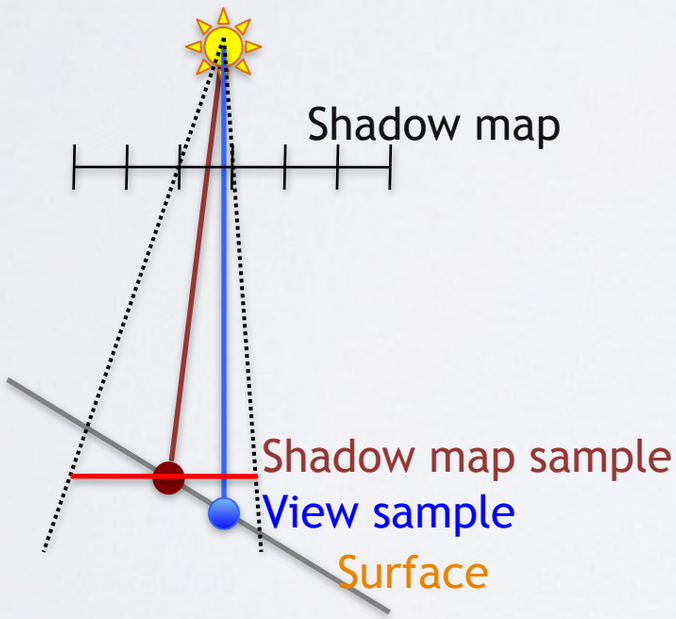
Shadow Maps - Problems

- In addition: A tolerance threshold(bias) needs to be tuned for each scene for the depth comparison



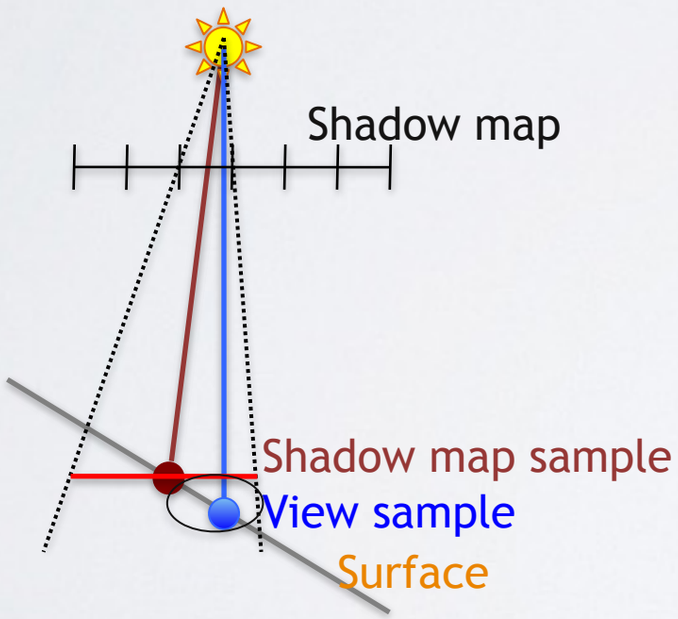
Shadow Maps - Problems

- In addition: A tolerance threshold(bias) needs to be tuned for each scene for the depth comparison



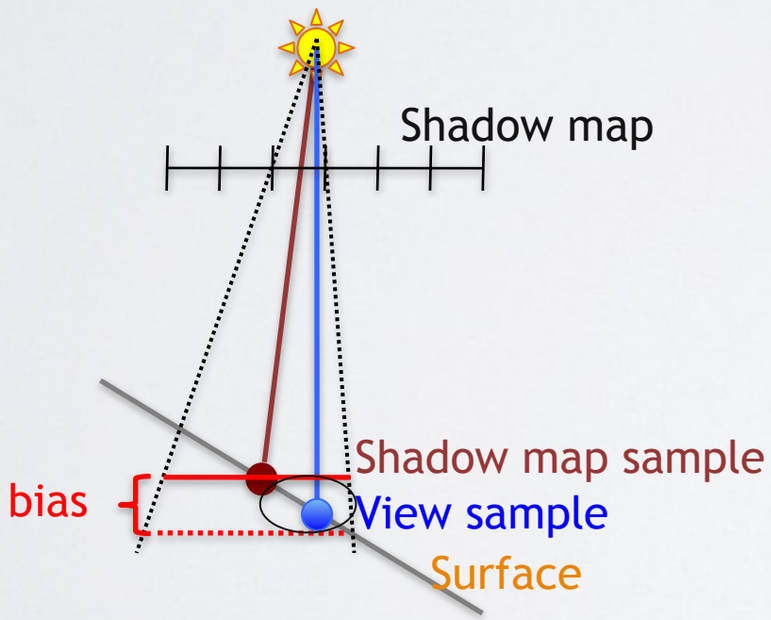
Shadow Maps - Problems

- In addition: A tolerance threshold(bias) needs to be tuned for each scene for the depth comparison



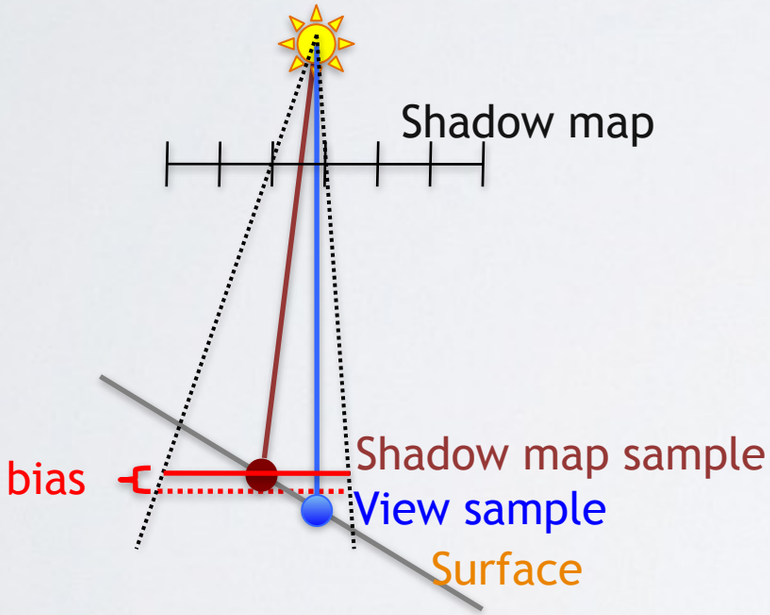
Shadow Maps - Problems

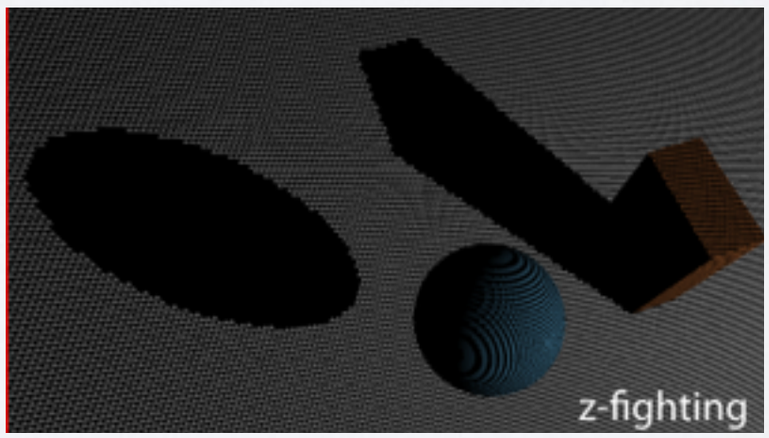
- In addition: A tolerance threshold(bias) needs to be tuned for each scene for the depth comparison



Shadow Maps - Problems

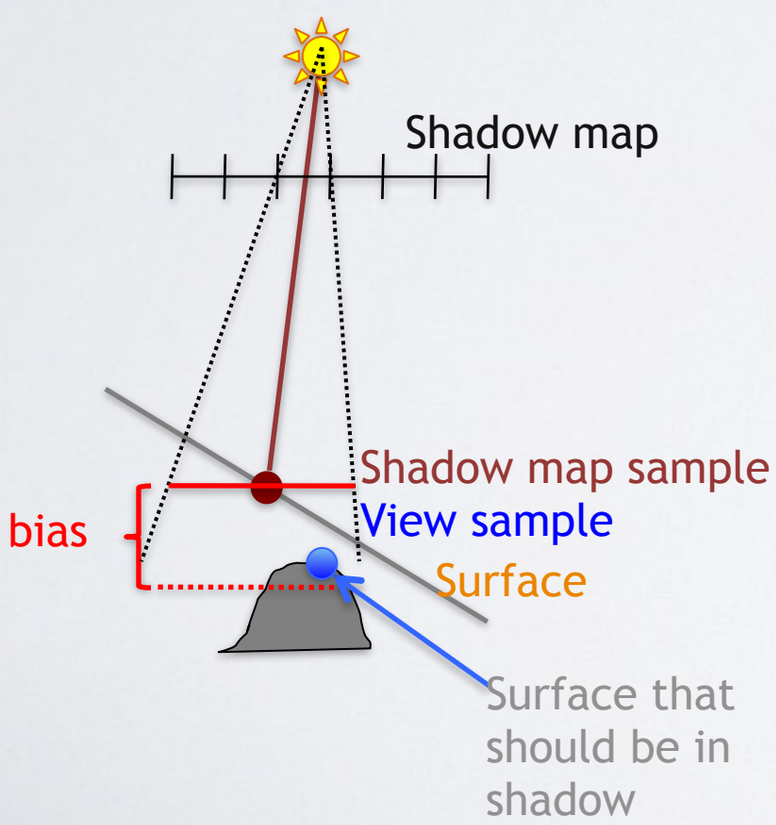
- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing

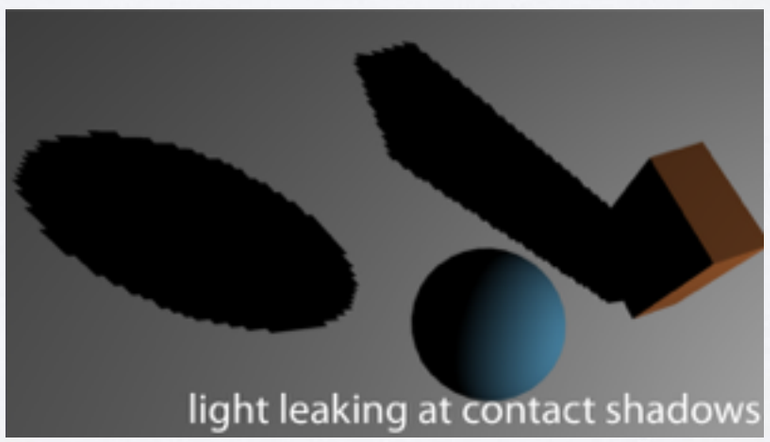




Shadow Maps - Problems

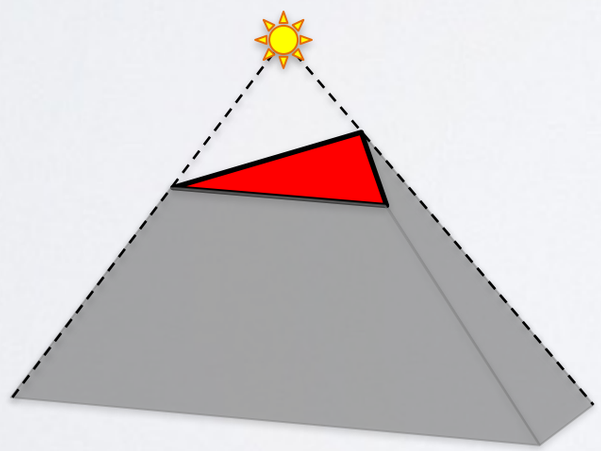
- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing





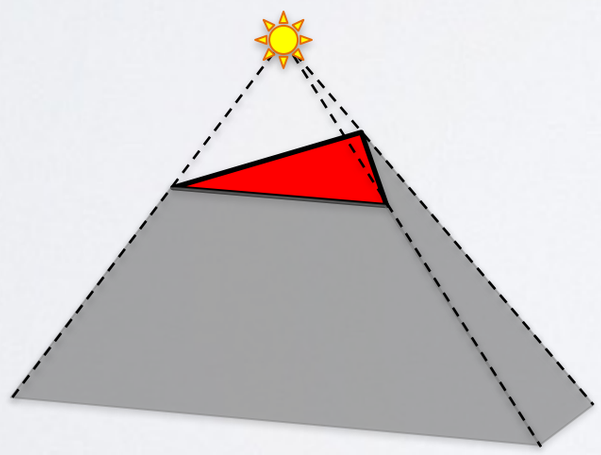
Shadow volumes

- Concept
    - Create volumes of "space in shadow" from each triangle
    - Each triangle creates 3 quads that extends to infinity



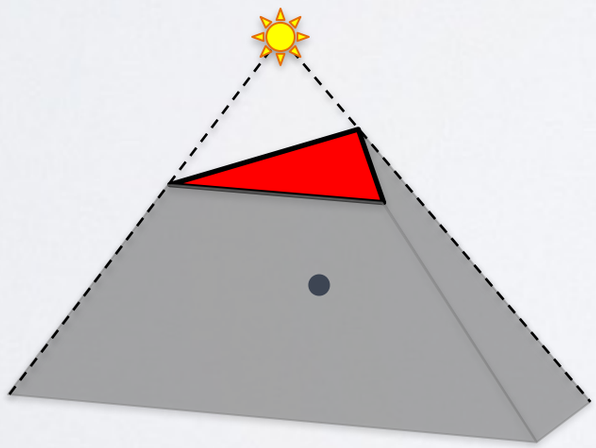
Shadow volumes

- Concept
    - Create volumes of "space in shadow" from each triangle
    - Each triangle creates 3 quads that extends to infinity



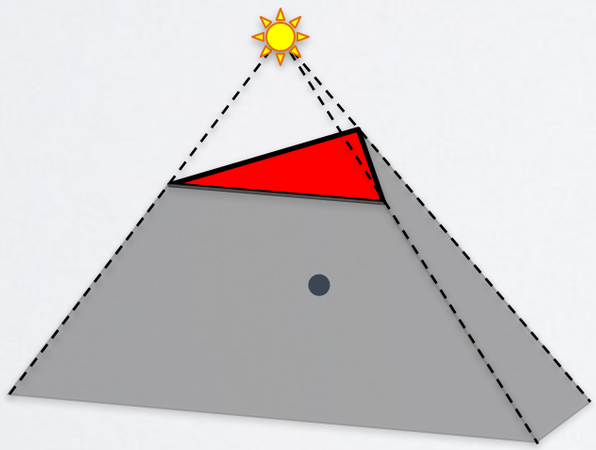
Shadow Volumes

- To test a point, count how many shadow volumes it is located within. One or more means point is in shadow



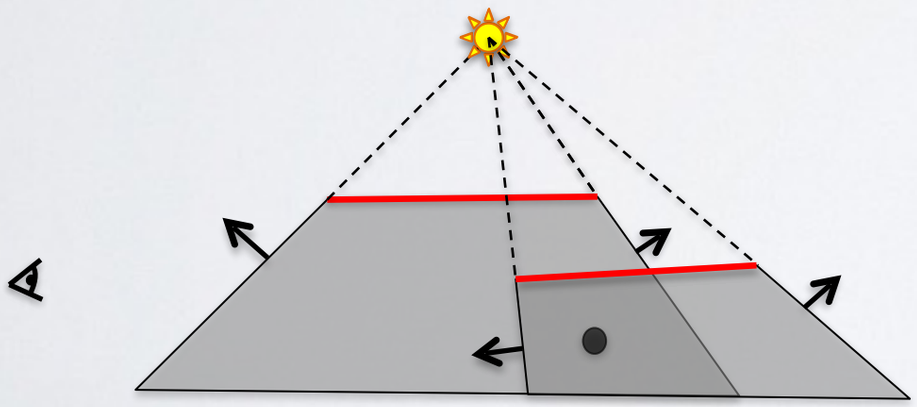
Shadow Volumes

- To test a point, count how many shadow volumes it is located within. One or more means point is in shadow



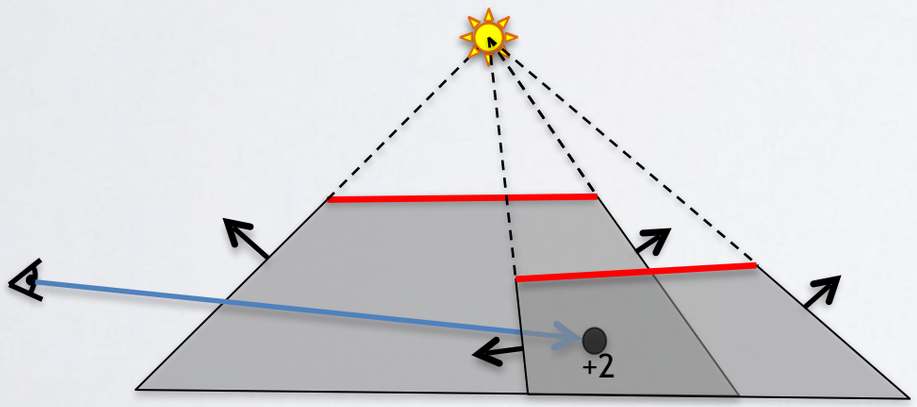
Shadow Volumes - concept

- A counter per pixel
- If we go through more frontfacing than backfacing polygons, then the point is in shadow



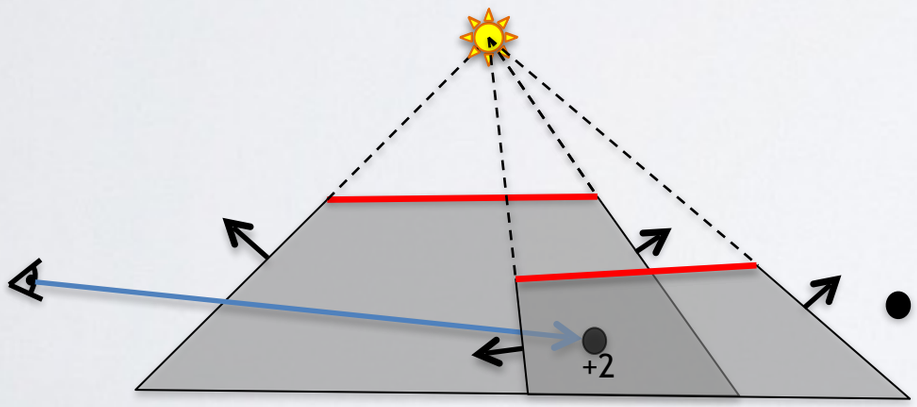
Shadow Volumes - concept

- A counter per pixel
- If we go through more frontfacing than backfacing polygons, then the point is in shadow



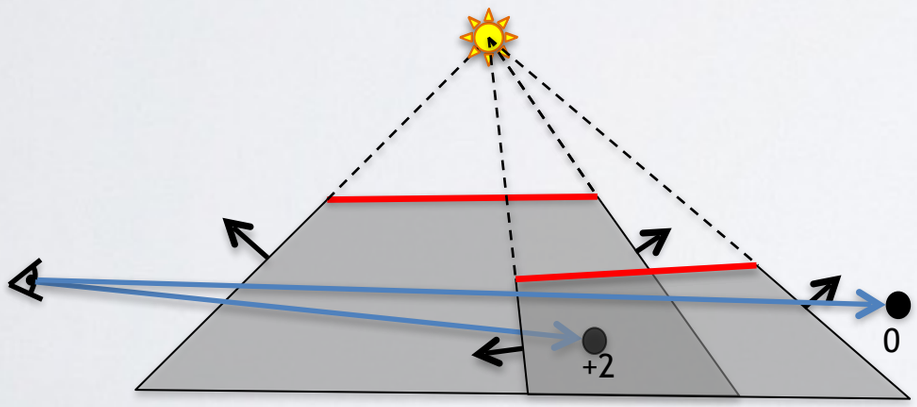
Shadow Volumes - concept

- A counter per pixel
- If we go through more frontfacing than backfacing polygons, then the point is in shadow



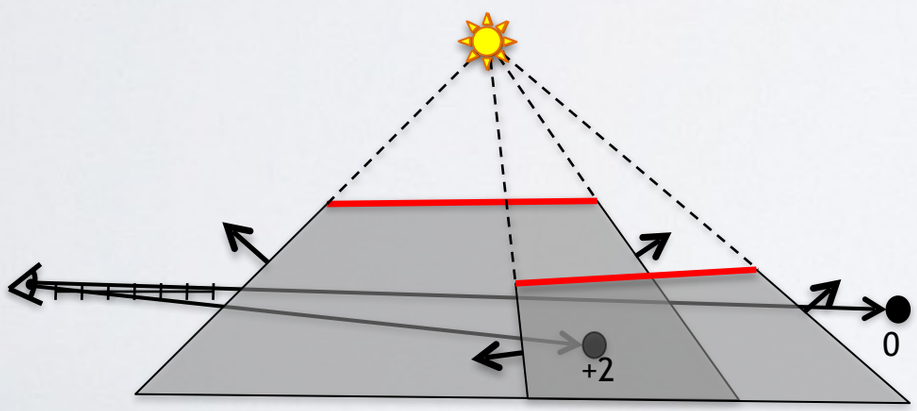
Shadow Volumes - concept

- A counter per pixel
- If we go through more frontfacing than backfacing polygons, then the point is in shadow



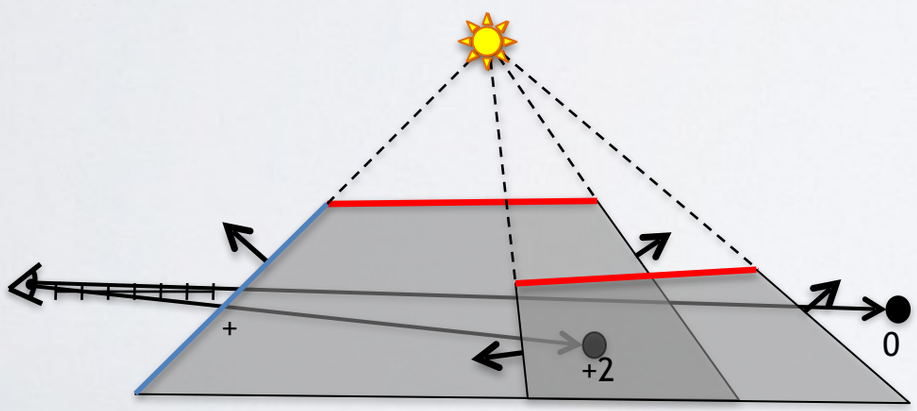
Shadow Volumes - concept

- Perform counting with the stencil buffer
    - Render front facing shadow quads to the stencil buffer (inc stencil value, since those represents entering shadow volume)
    - Render back facing shadow quads to the stencil buffer (dec stencil value, since those represents exiting shadow volume)



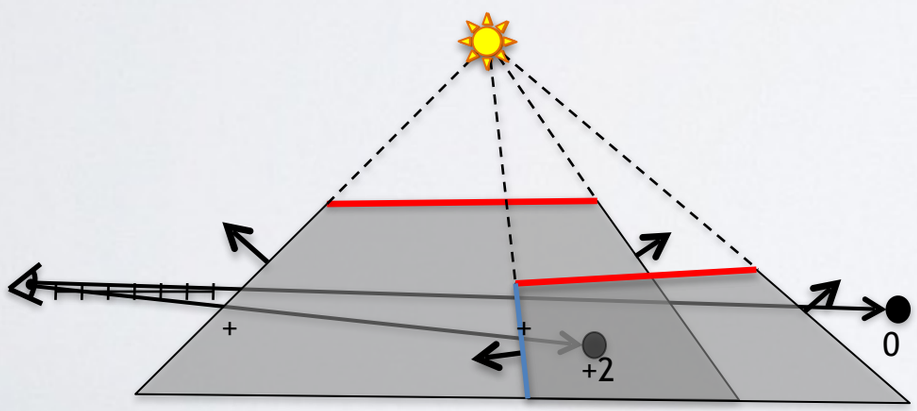
Shadow Volumes - concept

- Perform counting with the stencil buffer
    - Render front facing shadow quads to the stencil buffer (inc stencil value, since those represents entering shadow volume)
    - Render back facing shadow quads to the stencil buffer (dec stencil value, since those represents exiting shadow volume)



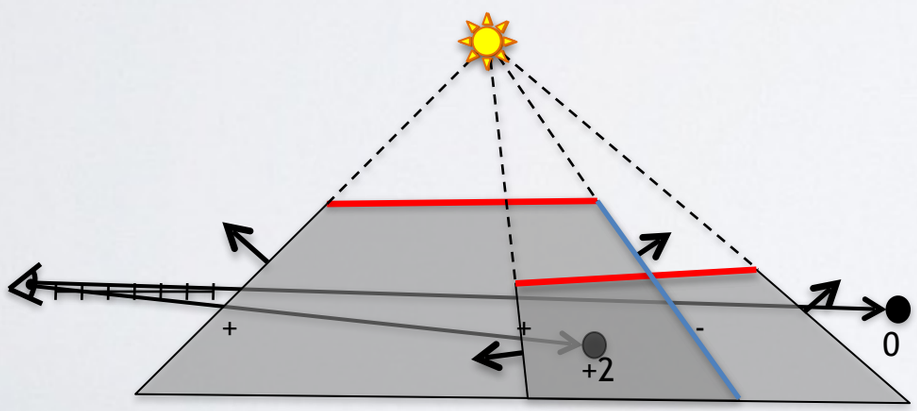
Shadow Volumes - concept

- Perform counting with the stencil buffer
    - Render front facing shadow quads to the stencil buffer (inc stencil value, since those represents entering shadow volume)
    - Render back facing shadow quads to the stencil buffer (dec stencil value, since those represents exiting shadow volume)



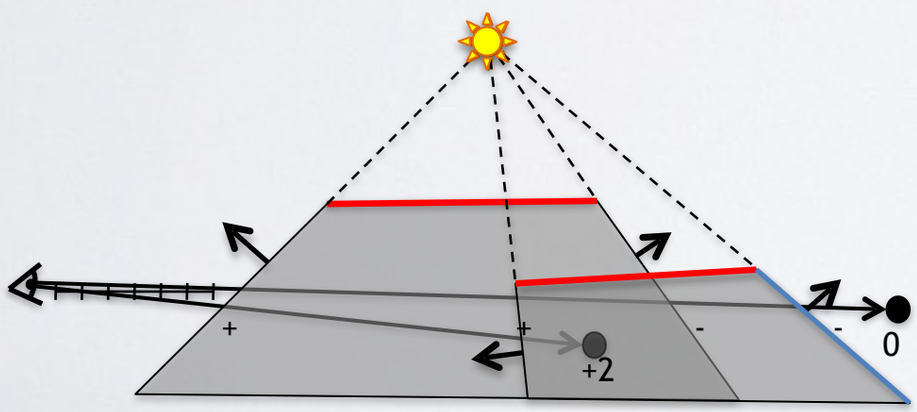
Shadow Volumes - concept

- Perform counting with the stencil buffer
    - Render front facing shadow quads to the stencil buffer (inc stencil value, since those represents entering shadow volume)
    - Render back facing shadow quads to the stencil buffer (dec stencil value, since those represents exiting shadow volume)



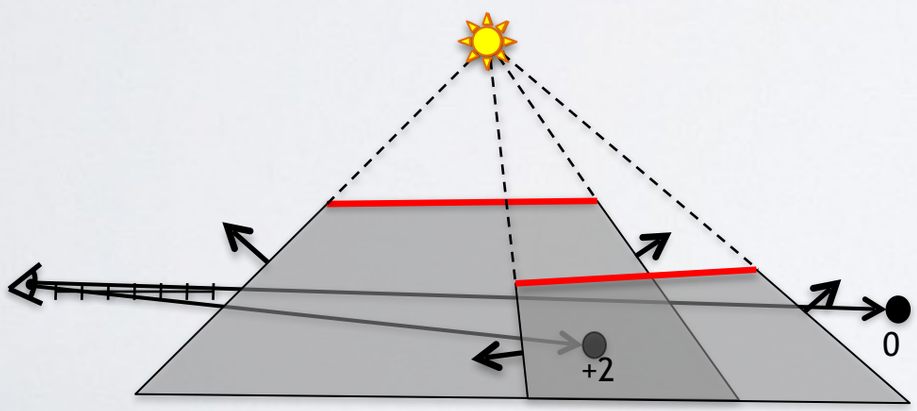
Shadow Volumes - concept

- Perform counting with the stencil buffer
    - Render front facing shadow quads to the stencil buffer (inc stencil value, since those represents entering shadow volume)
    - Render back facing shadow quads to the stencil buffer (dec stencil value, since those represents exiting shadow volume)



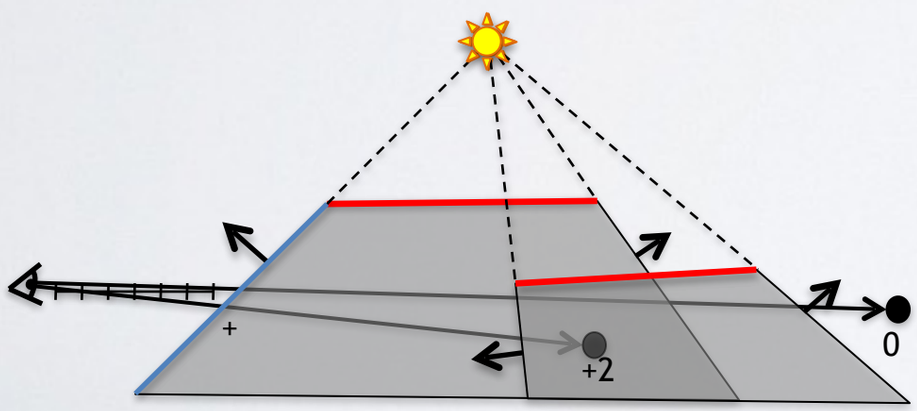
Shadow Volumes - concept

- Perform counting with the stencil buffer
    - Render front facing shadow quads to the stencil buffer (inc stencil value, since those represents entering shadow volume)
    - Render back facing shadow quads to the stencil buffer (dec stencil value, since those represents exiting shadow volume)



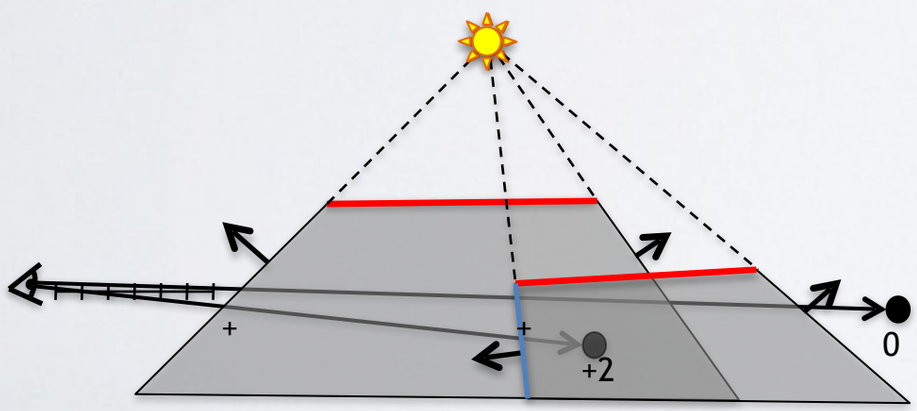
Shadow Volumes - concept

- Perform counting with the stencil buffer
    - Render front facing shadow quads to the stencil buffer (inc stencil value, since those represents entering shadow volume)
    - Render back facing shadow quads to the stencil buffer (dec stencil value, since those represents exiting shadow volume)



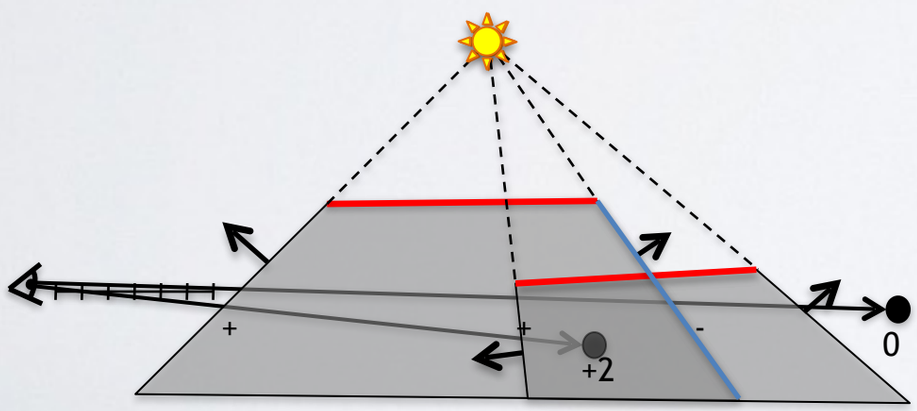
Shadow Volumes - concept

- Perform counting with the stencil buffer
    - Render front facing shadow quads to the stencil buffer (inc stencil value, since those represents entering shadow volume)
    - Render back facing shadow quads to the stencil buffer (dec stencil value, since those represents exiting shadow volume)



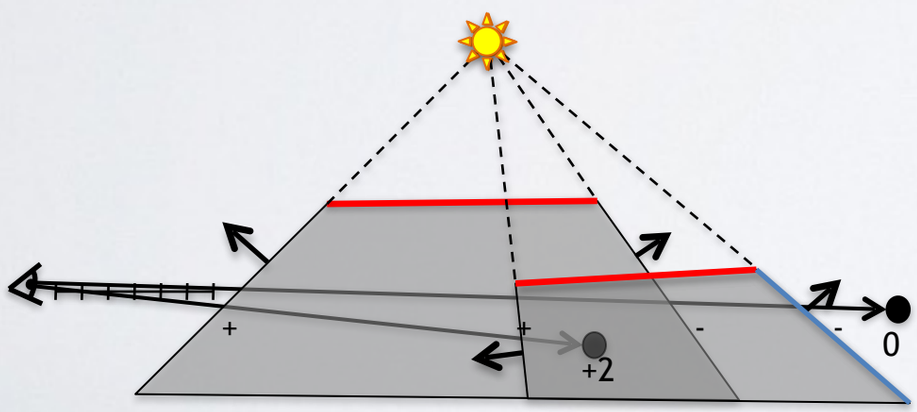
Shadow Volumes - concept

- Perform counting with the stencil buffer
    - Render front facing shadow quads to the stencil buffer (inc stencil value, since those represents entering shadow volume)
    - Render back facing shadow quads to the stencil buffer (dec stencil value, since those represents exiting shadow volume)



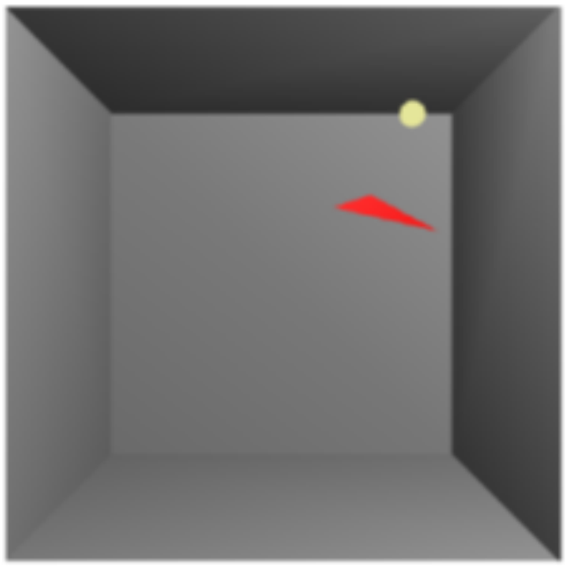
Shadow Volumes - concept

- Perform counting with the stencil buffer
    - Render front facing shadow quads to the stencil buffer (inc stencil value, since those represents entering shadow volume)
    - Render back facing shadow quads to the stencil buffer (dec stencil value, since those represents exiting shadow volume)

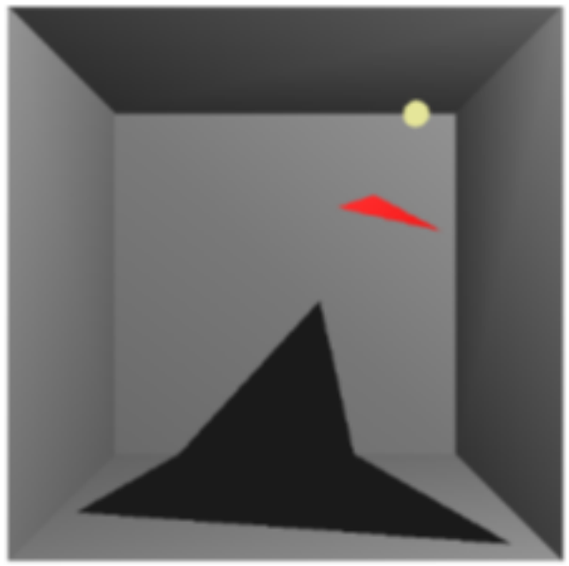


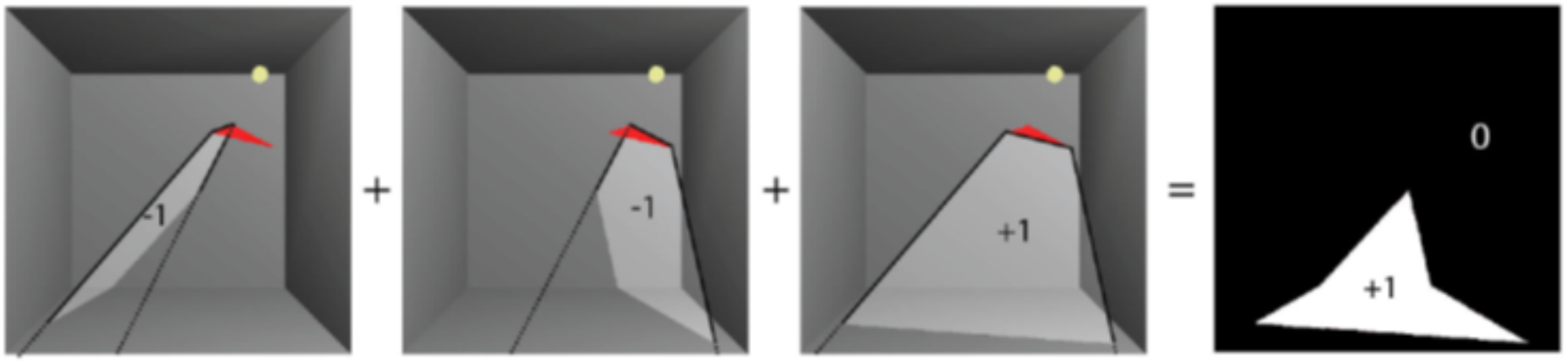
Z-pass by example: how the stencil buffer is used

What we have...



What we wnat...



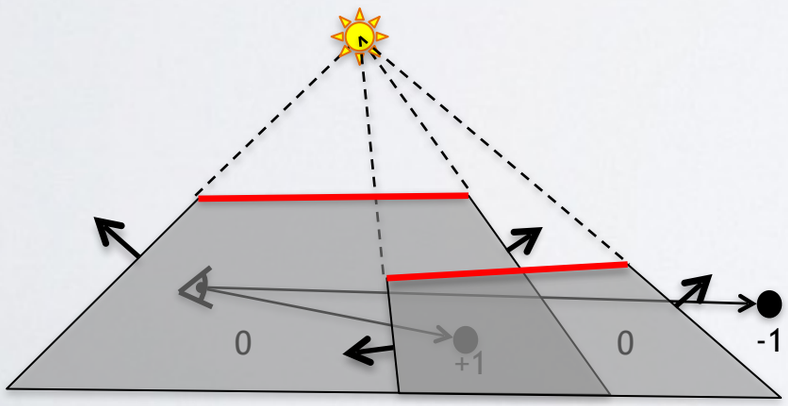


Shadow Volumes with the Stencil Buffer

- A three pass process
    - pass 1: Render ambient lighting;
    - pass 2: Draw to stencil buffer only
        - Turn off updating of z-buffer and writing to color buffer but still use standard depth test
        - Set stencil operation to
            - incrementing stencil buffer count for frontfacing shadow volume quads, and
            - decrementing stencil buffer count for backfacing shadow volumes quads
- pass 3: Render diffuse and specular where stencil buffer is 0.

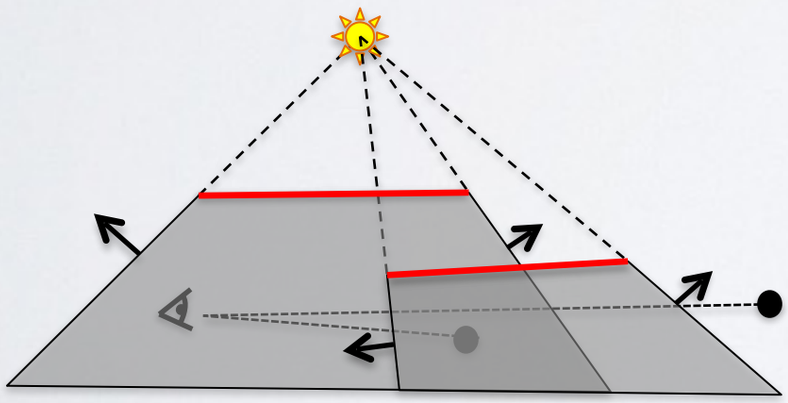
Eye Location Problem

- If the eye is located inside one or more shadow volumes, then the count will be wrong
- Solution:
    - Offset stencil buffer with the shadow volumes that the eye is located within
    - Or modify the way we do the counting...



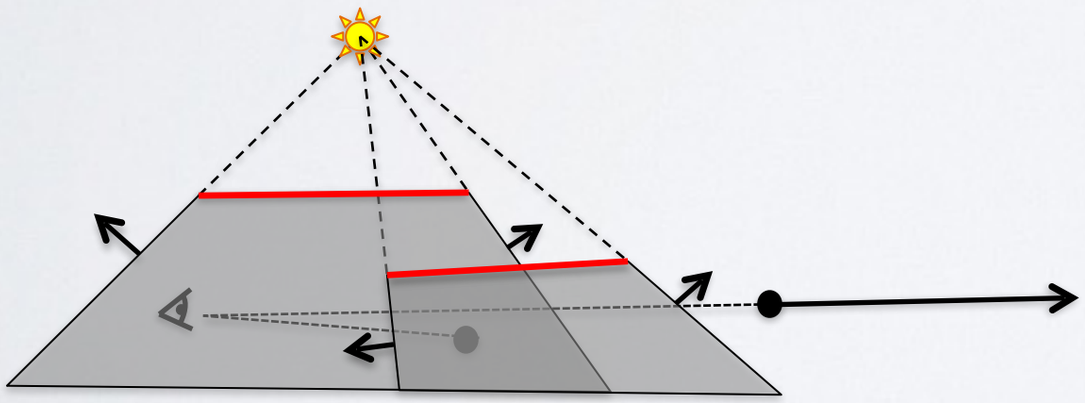
The Z-fail Algorithm ("Carmacks Reverse")

- Count to infinity instead of to the eye
    - We can choose any reference location for the counting
    - A point in light avoids any offset
    - Infinity is always in light - if we cap the shadow volumes at infinity



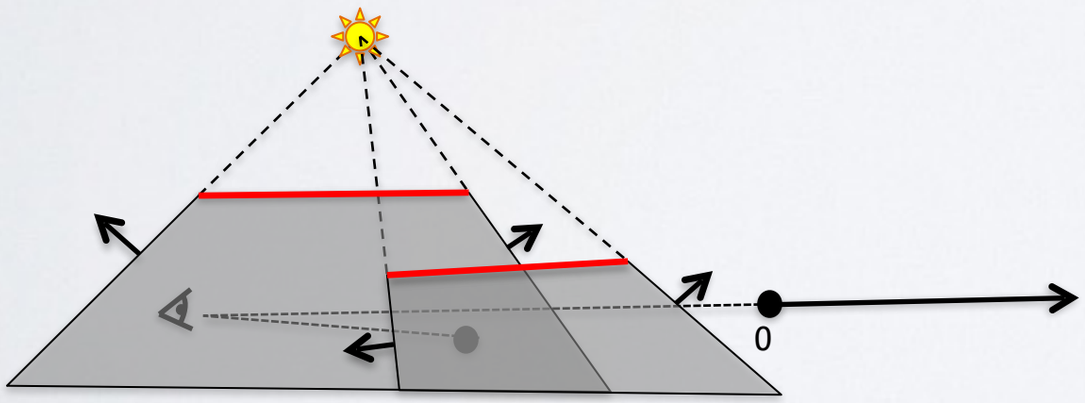
The Z-fail Algorithm ("Carmacks Reverse")

- Count to infinity instead of to the eye
    - We can choose any reference location for the counting
    - A point in light avoids any offset
    - Infinity is always in light - if we cap the shadow volumes at infinity



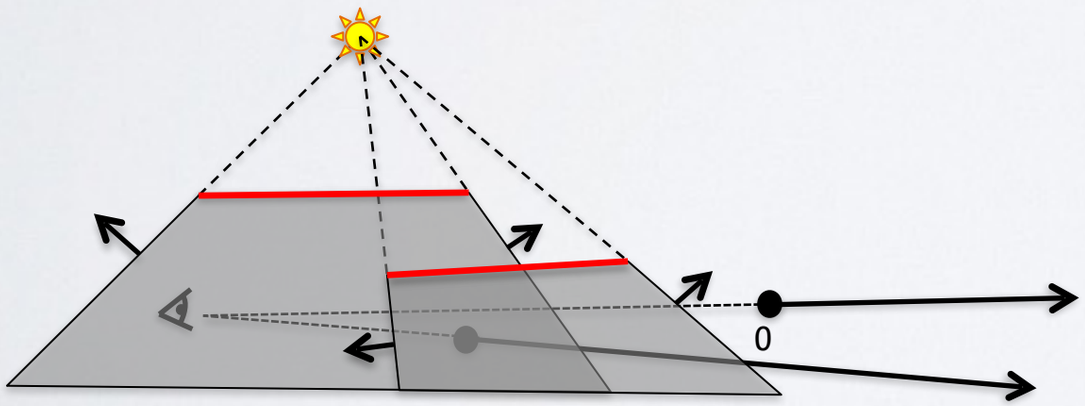
The Z-fail Algorithm ("Carmacks Reverse")

- Count to infinity instead of to the eye
    - We can choose any reference location for the counting
    - A point in light avoids any offset
    - Infinity is always in light - if we cap the shadow volumes at infinity



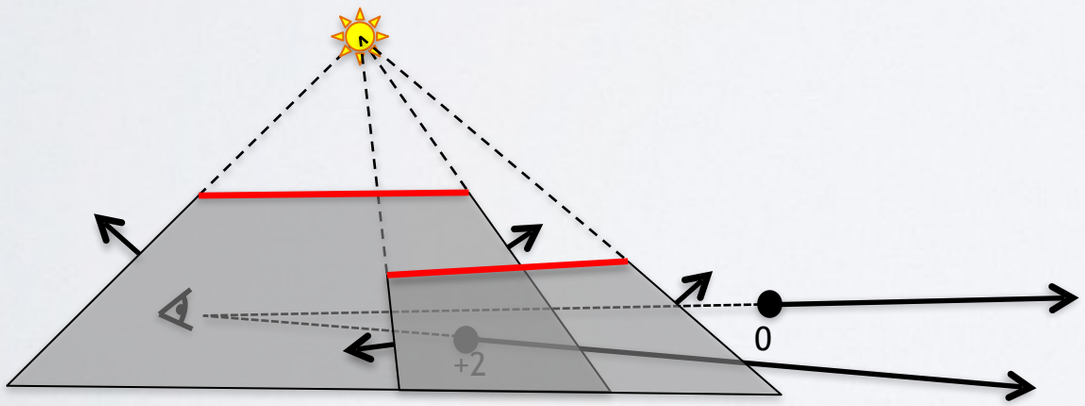
The Z-fail Algorithm ("Carmacks Reverse")

- Count to infinity instead of to the eye
    - We can choose any reference location for the counting
    - A point in light avoids any offset
    - Infinity is always in light - if we cap the shadow volumes at infinity



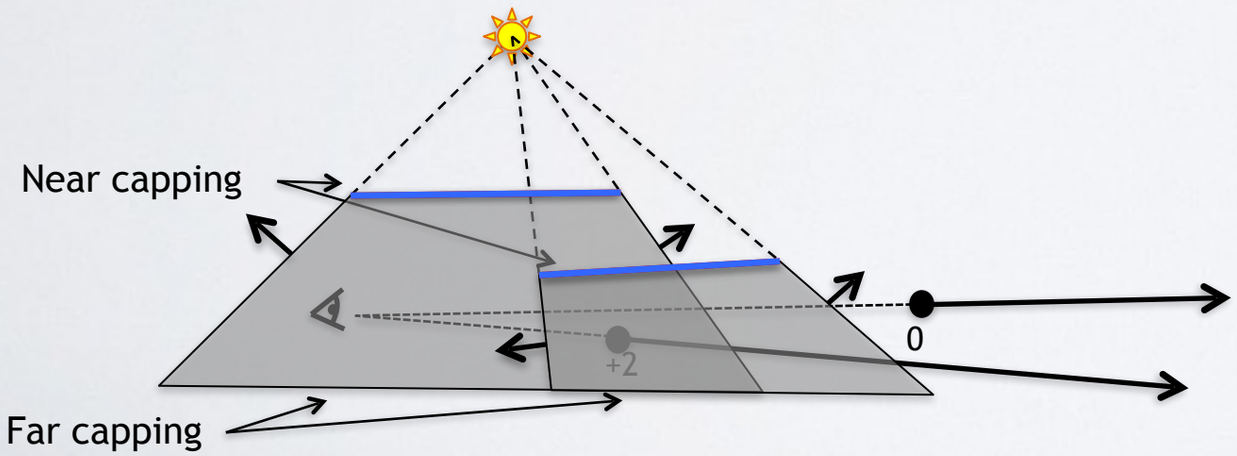
The Z-fail Algorithm ("Carmacks Reverse")

- Count to infinity instead of to the eye
    - We can choose any reference location for the counting
    - A point in light avoids any offset
    - Infinity is always in light - if we cap the shadow volumes at infinity

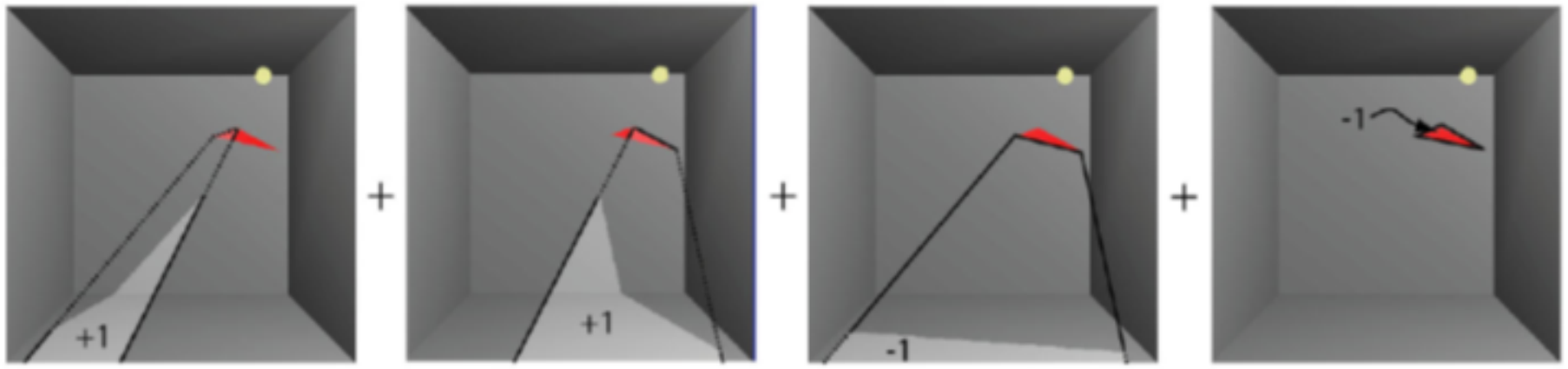


The Z-fail Algorithm ("Carmacks Reverse")

- Count to infinity instead of to the eye
    - We can choose any reference location for the counting
    - A point in light avoids any offset
    - Infinity is always in light - if we cap the shadow volumes at infinity



Z-fail by example

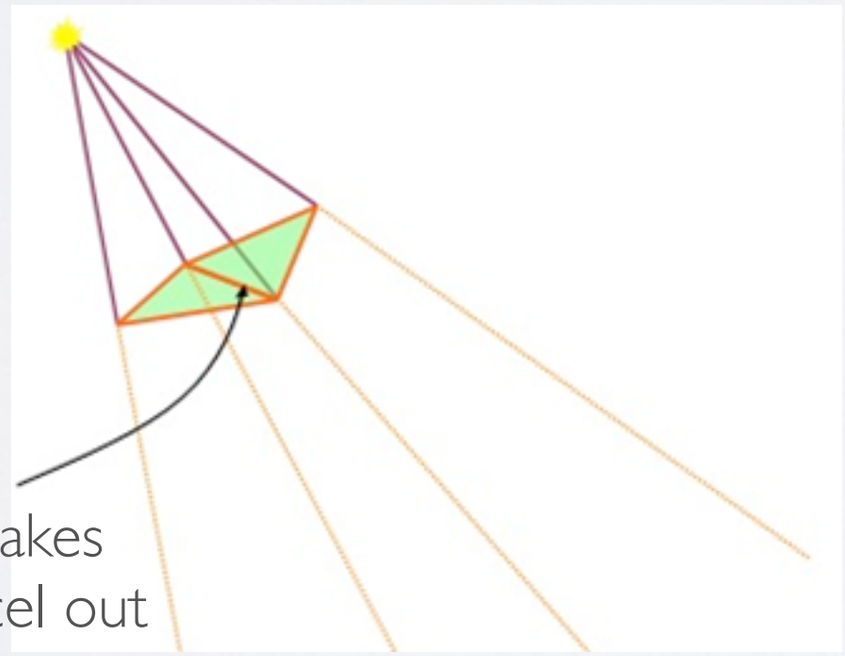


Compare to Z-pass:

- Invert z-test
- Invert stencil inc/dec

Shadow Volumes from Silhouette Edges

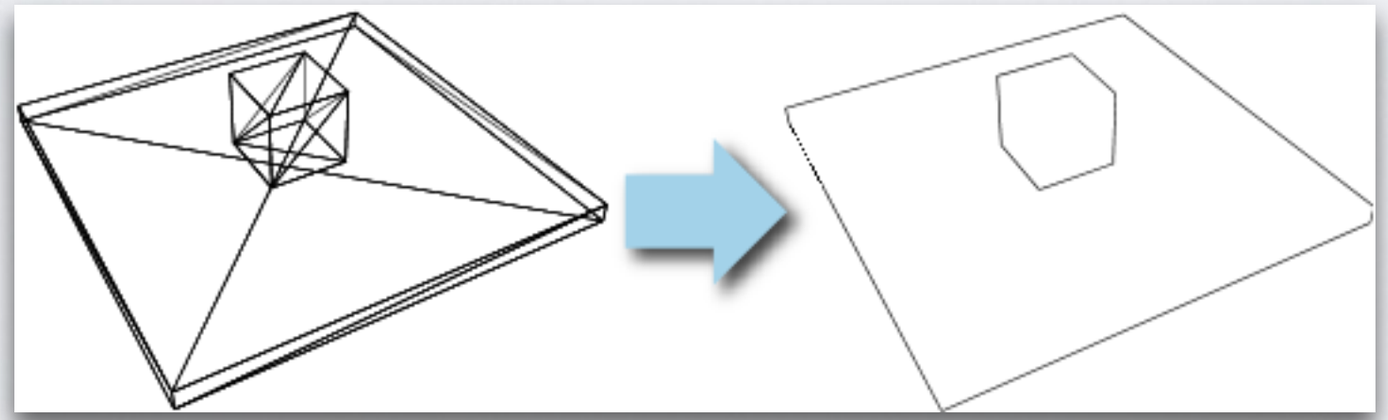
- Merging shadow volumes:
    - An interior edge(non-silhouette edge as seen from the light position) creates two shadow quads that cancel each other out:
    - Thus, create shadow volumes only from the silhouette edges. (avoid rendering of many useless shadow quads)



This interior edge makes two quads, which cancel out

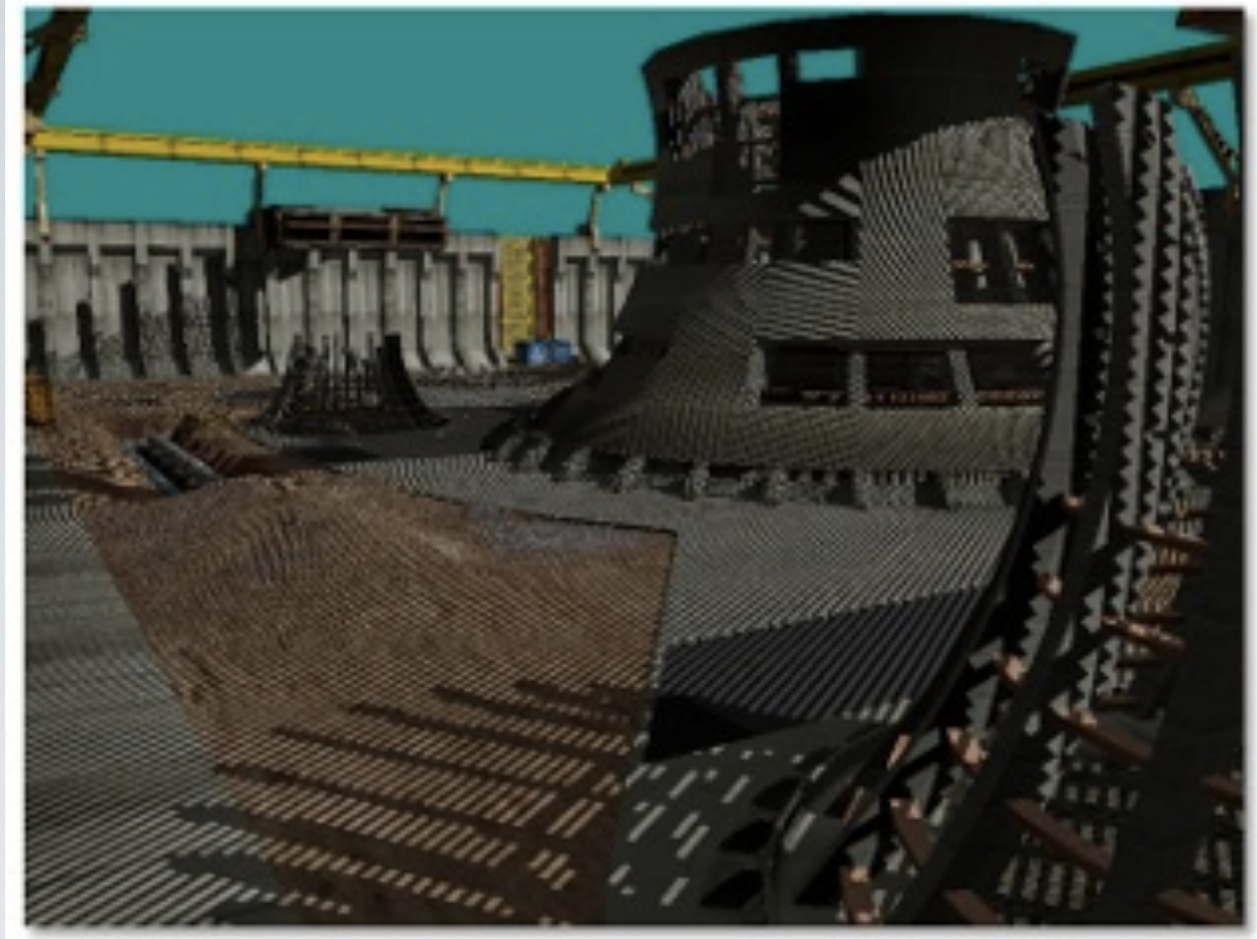
Shadow Volumes from Silhouette Edges

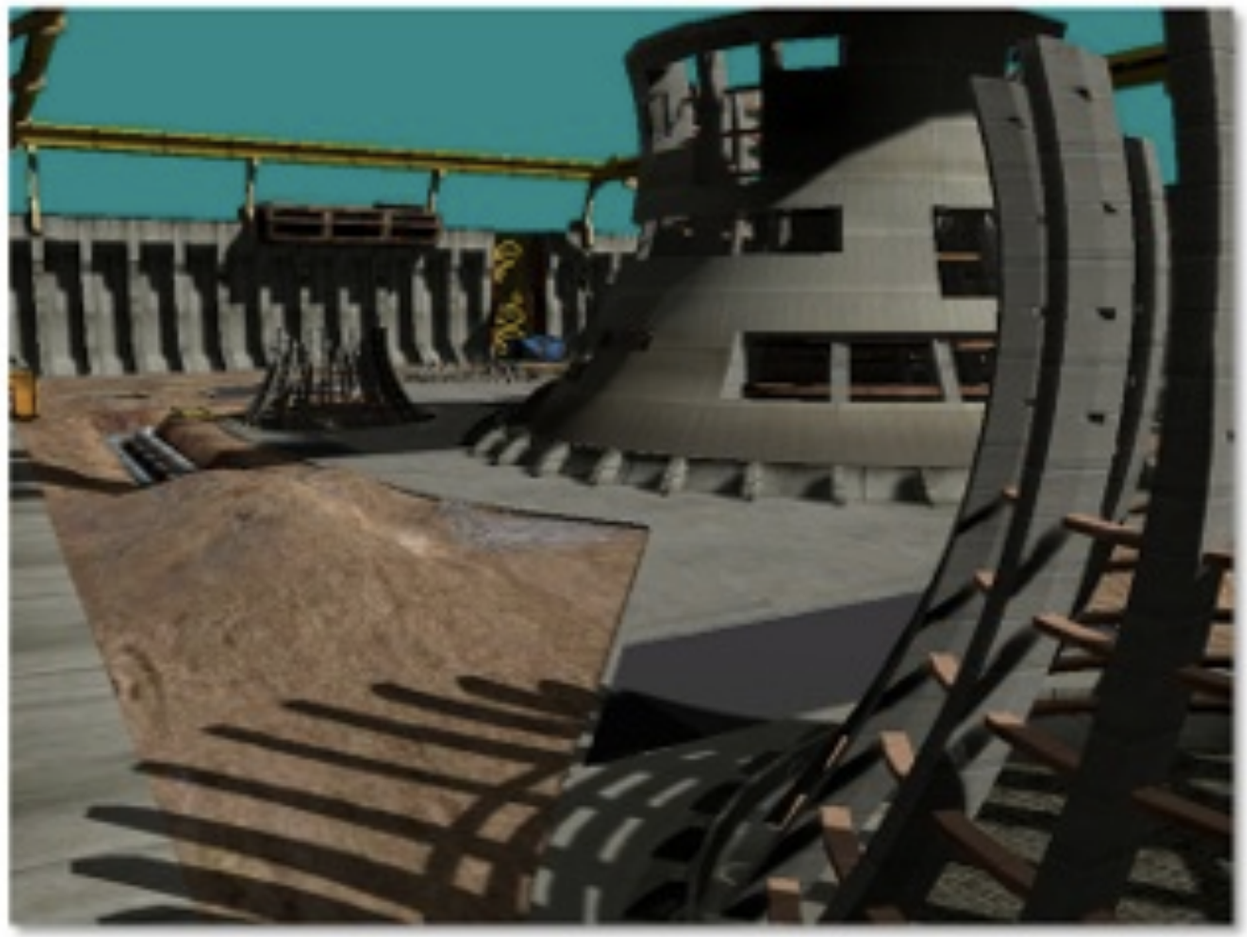
A real example:



Shadow Volumes - Summary

- Pros:
    - High quality
- Cons:
    - Overdraw

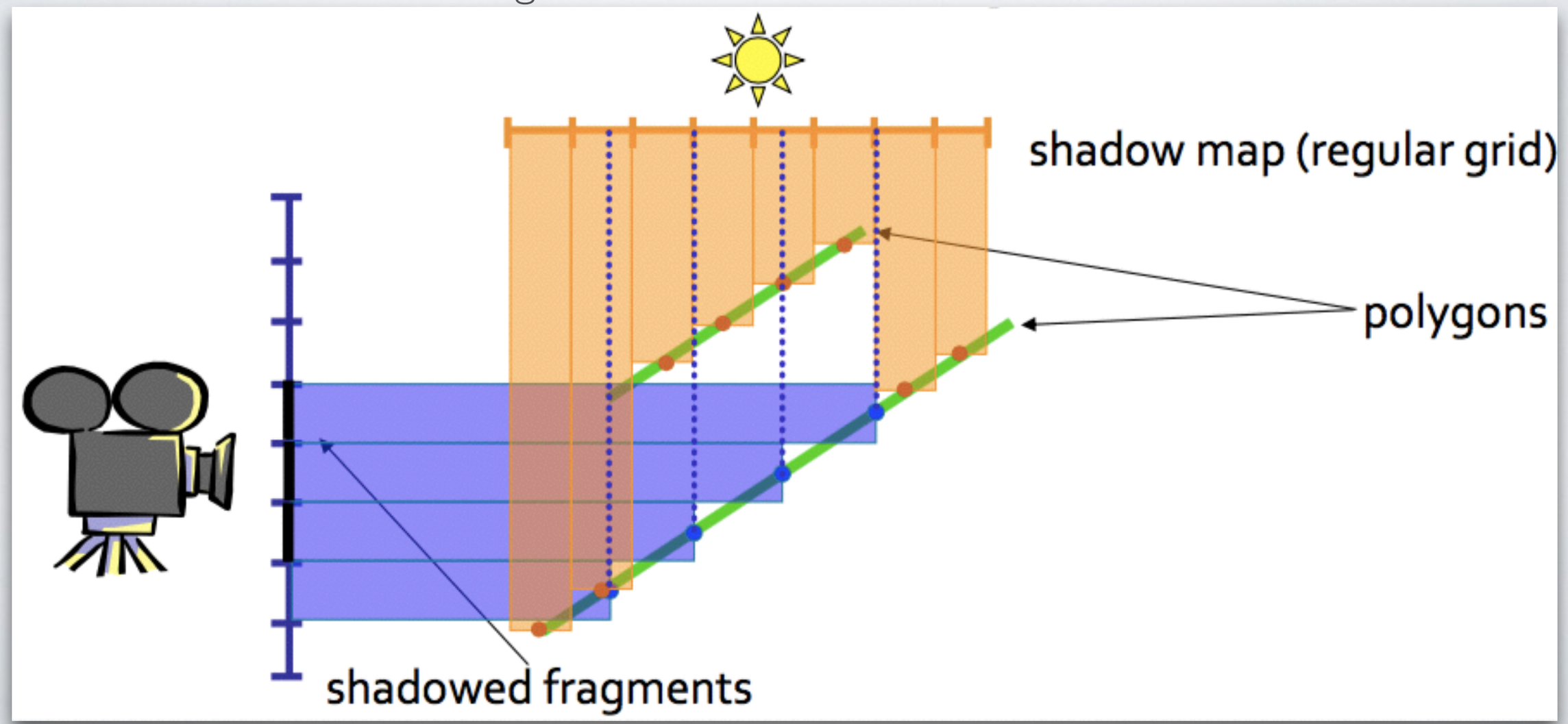




SHADOW MAP ANTI-ALIASING

Shadow Map as Signal Reconstruction

- Initial sampling: shadow-map rendering
- Resampling: determined by view
- Reconstruction: nearest neighbor, PCF, …

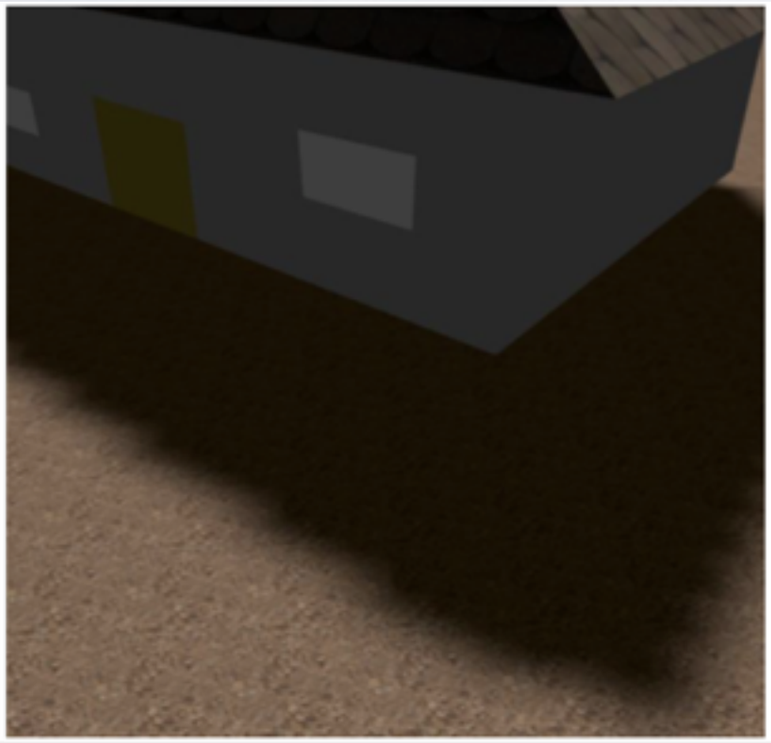


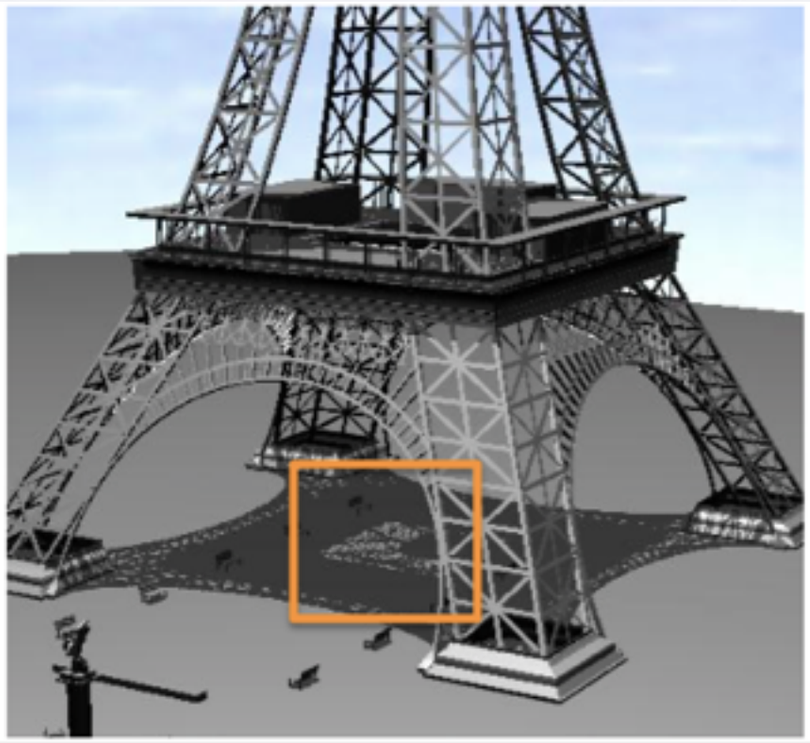
Main Types of Error

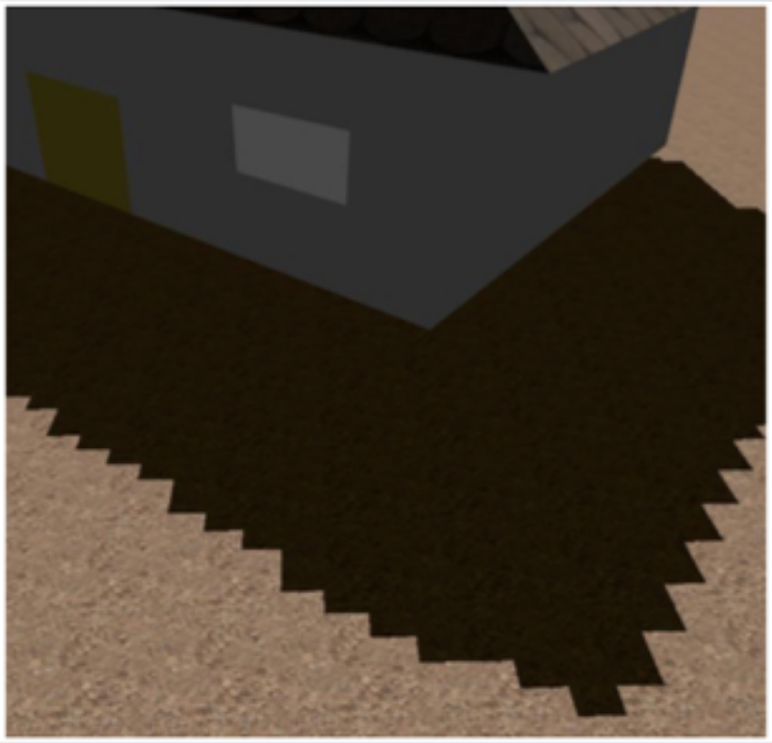
Initial sampling:
Undersampling

Resampling:
Oversampling

Reconstruction:
Reconstruction Error

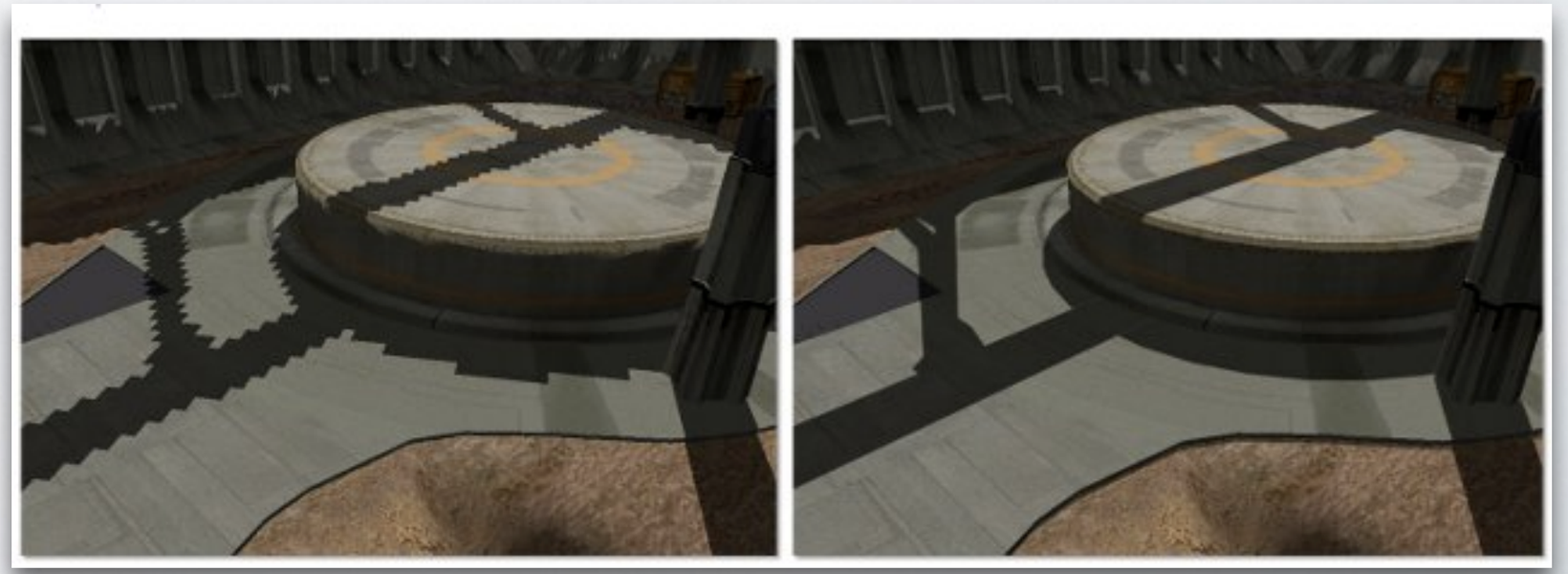






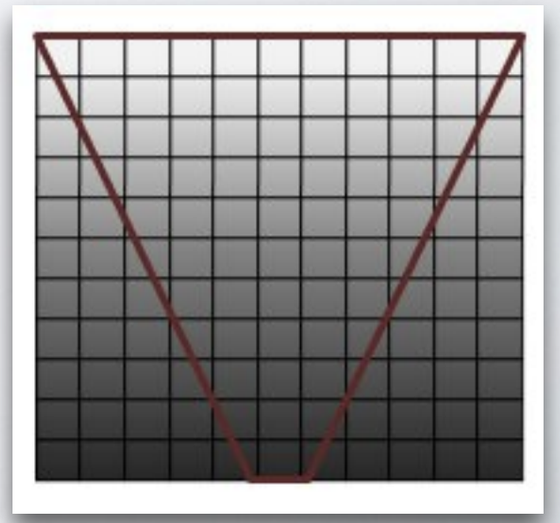
Shadow Map Artifacts

- Perspective Aliasing
    - occurs when the mapping of pixels in view space to texels in the shadow map is not a one-to-one ratio



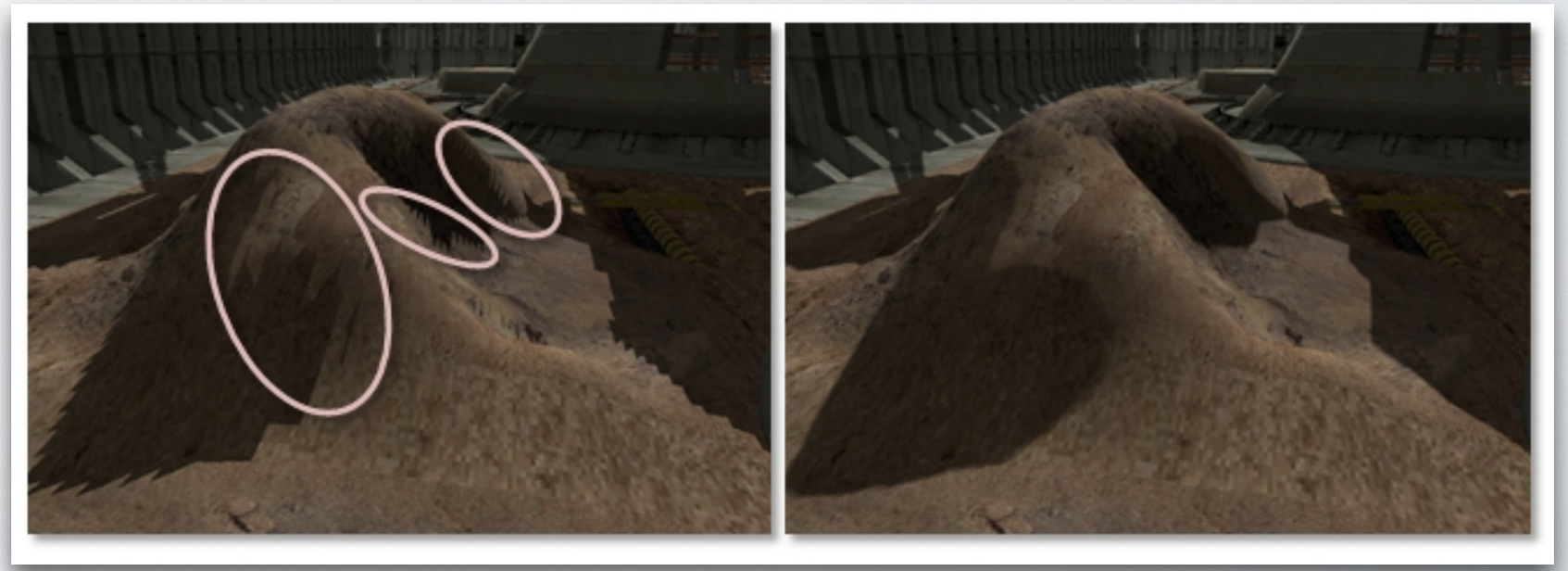
Perspective Aliasing

- View frustum with shadow map
    - Near the eye, the pixels are closer together, and many pixels map to the same shadow texels;
    - Light pixels in the far plane represent low-perspective aliasing, and dark pixels in the near plane represent high-perspective aliasing.



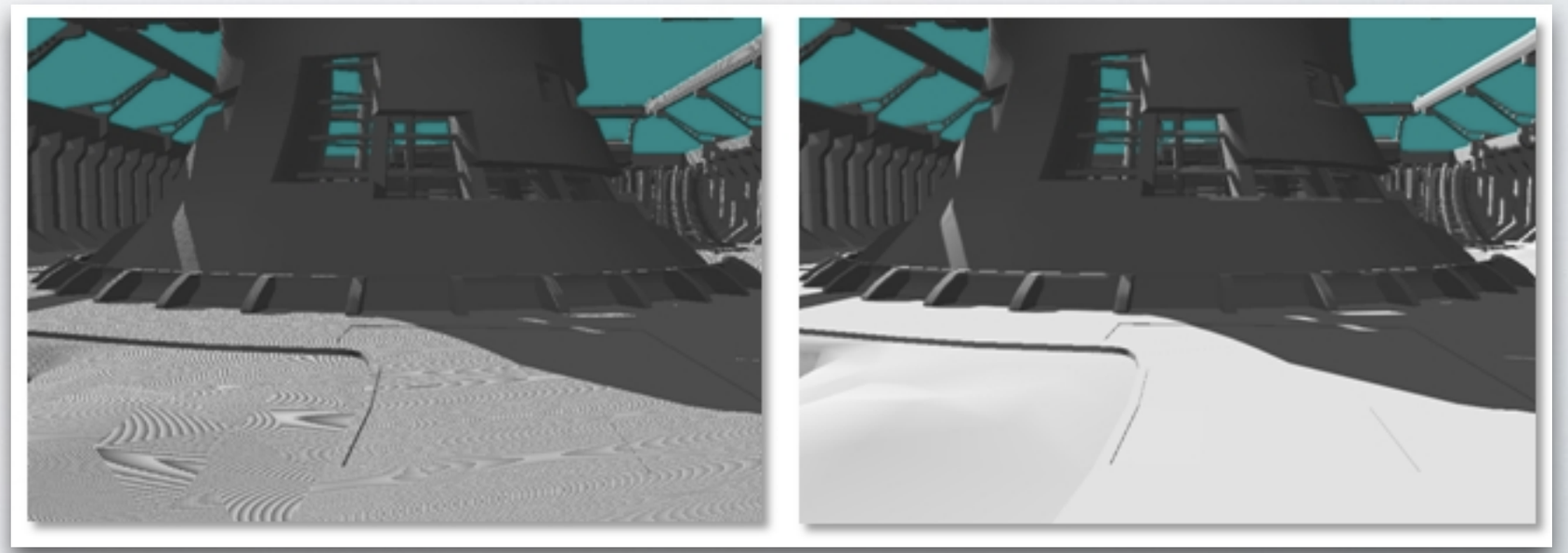
Projective Alising

- Projective aliasing occurs as the tangent plane of the geometry becomes parallel to the light rays;
- Techniques used to alleviate perspective aliasing errors also mitigate projective aliasing.



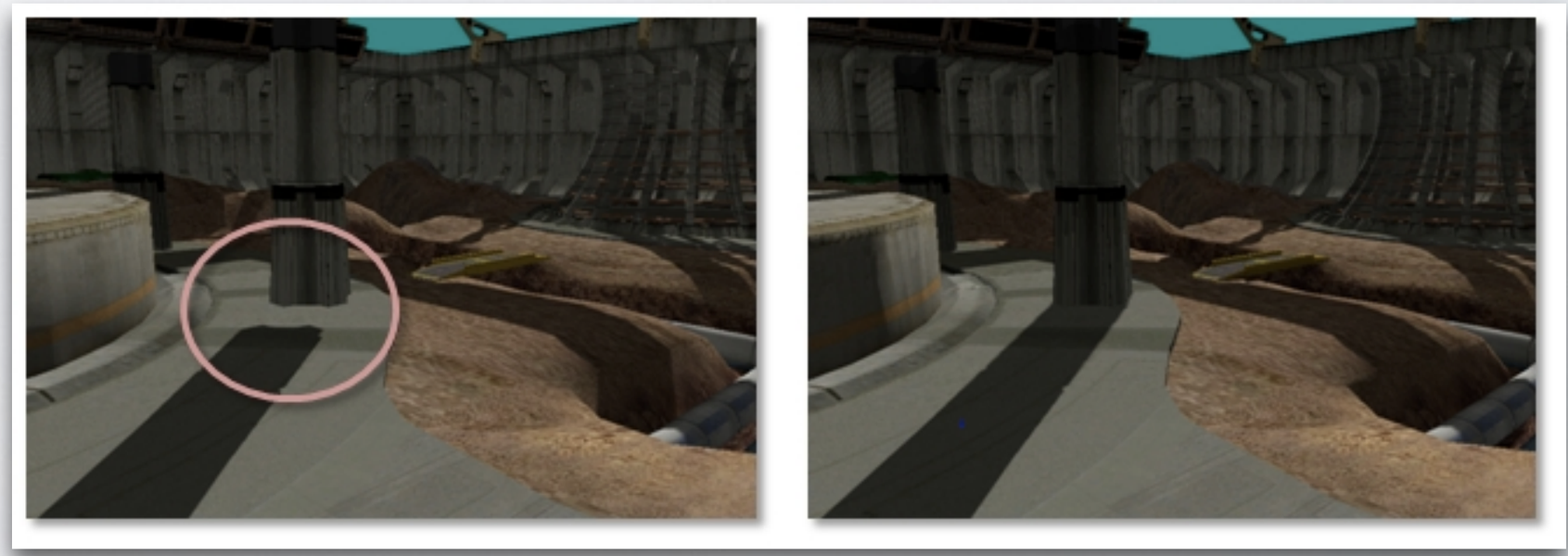
Shadow Acne and Erroneous Self-Shadowing

- some of pixels failed the depth test and created speckled artifacts and moiré patterns;
- when the texel in light space is so close to the depth of the corresponding texel in the depth map that precision errors cause the depth test to erroneously fail



Light bleeding (Peter Panning)

- the shadow is detached from the object, creating a floating effect;



Techniques to Improve Shadow Maps

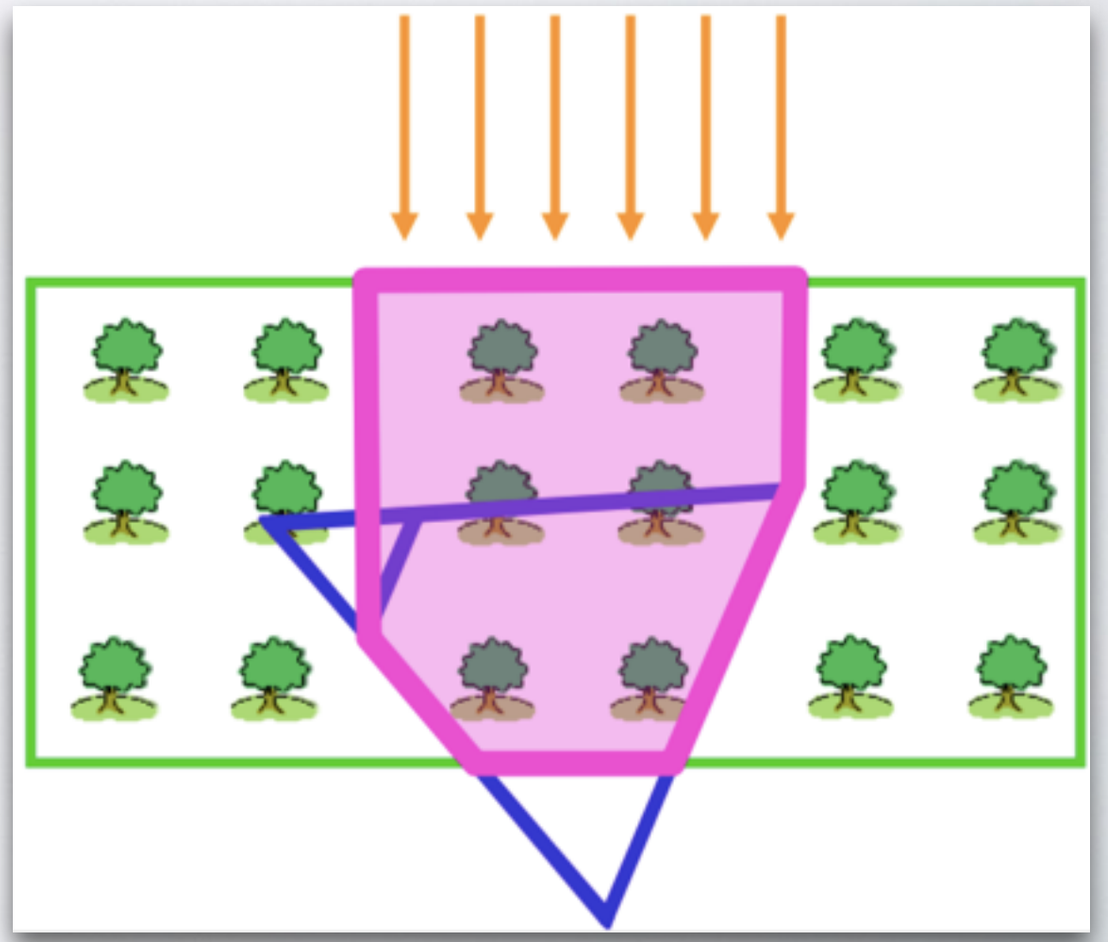
- Undersampling:
    - Improve initial sampling
- Oversampling:
    - Use bandlimiting filter in reconstruction(VSM, PCF)

Improve Initial Sampling

- Fitting (Focusing)
- Warping
- Partitioning

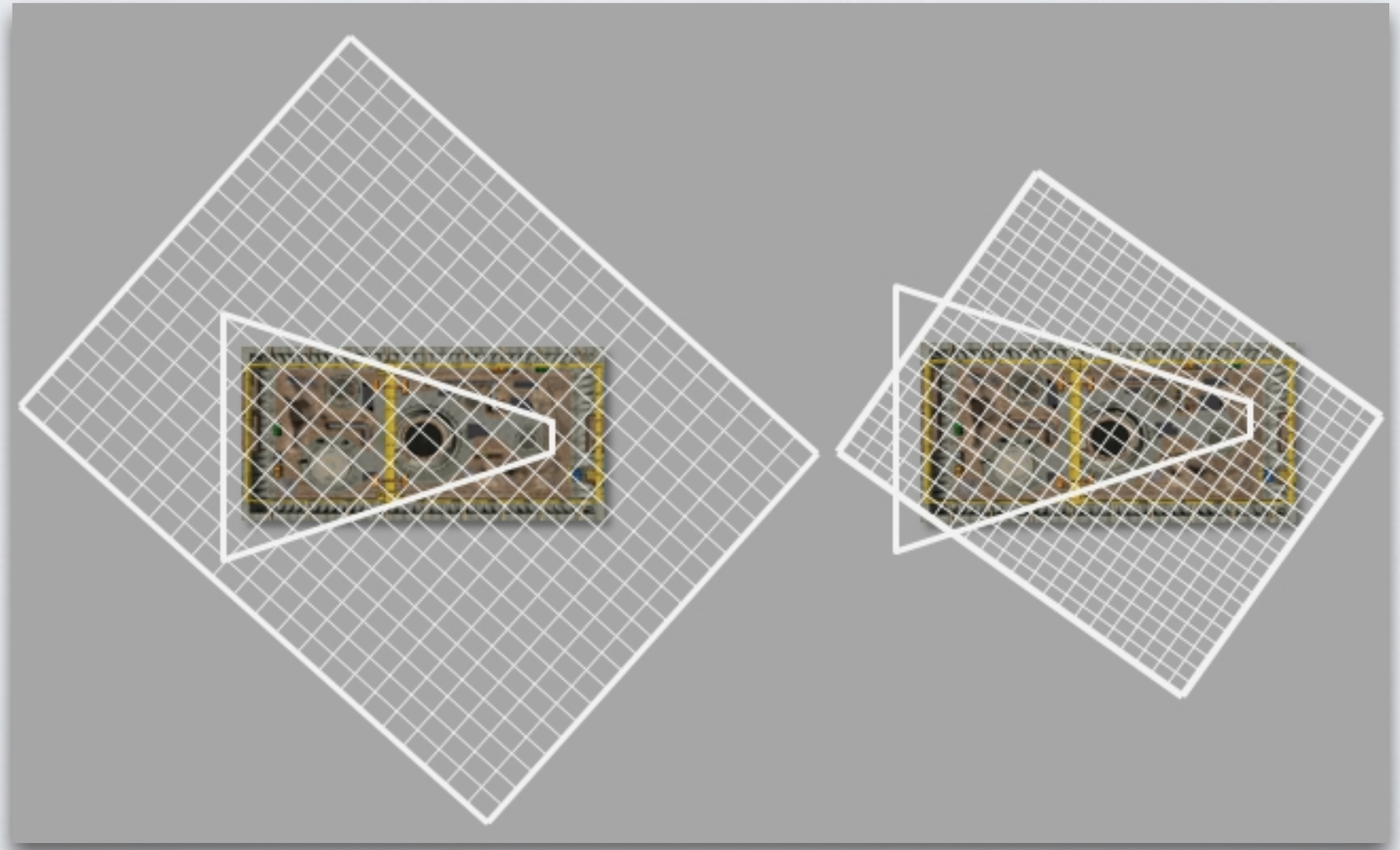
Fitting: Focus the Shadow Map

- Only include relevant objects;
- Intersection body determined by:
  - Shadow casters;
  - Light source frustum;
  - View frustum



Fitting Calculation

- The image on the right shows that the same resolution shadow map creates more texel coverage when it is fit more tightly to the scene.

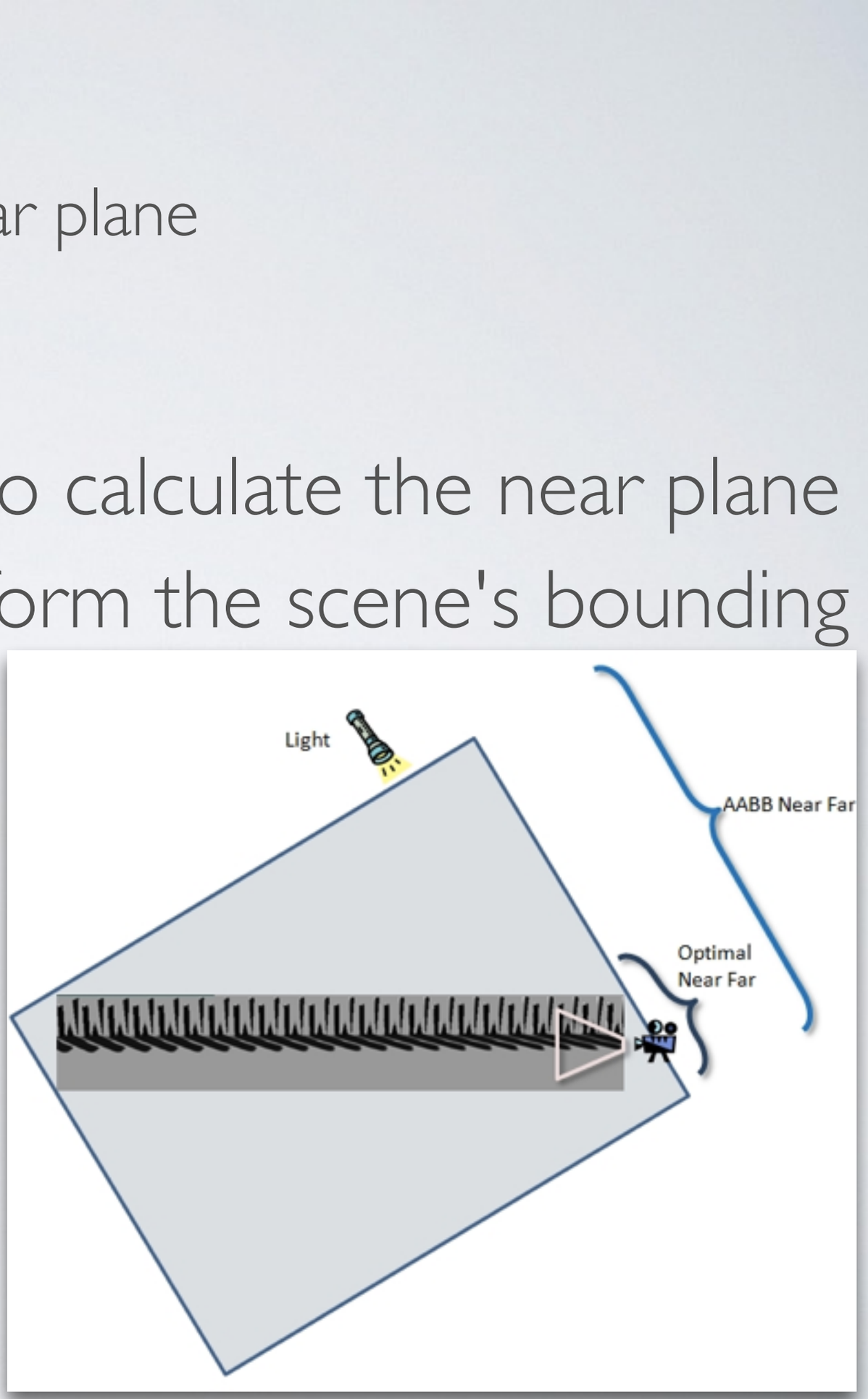


Calculating the near plane and far plane

- The near plane and far plane are the final pieces required to calculate the projection matrix. The more closely together the planes are, the more precise the values in the depth buffer.

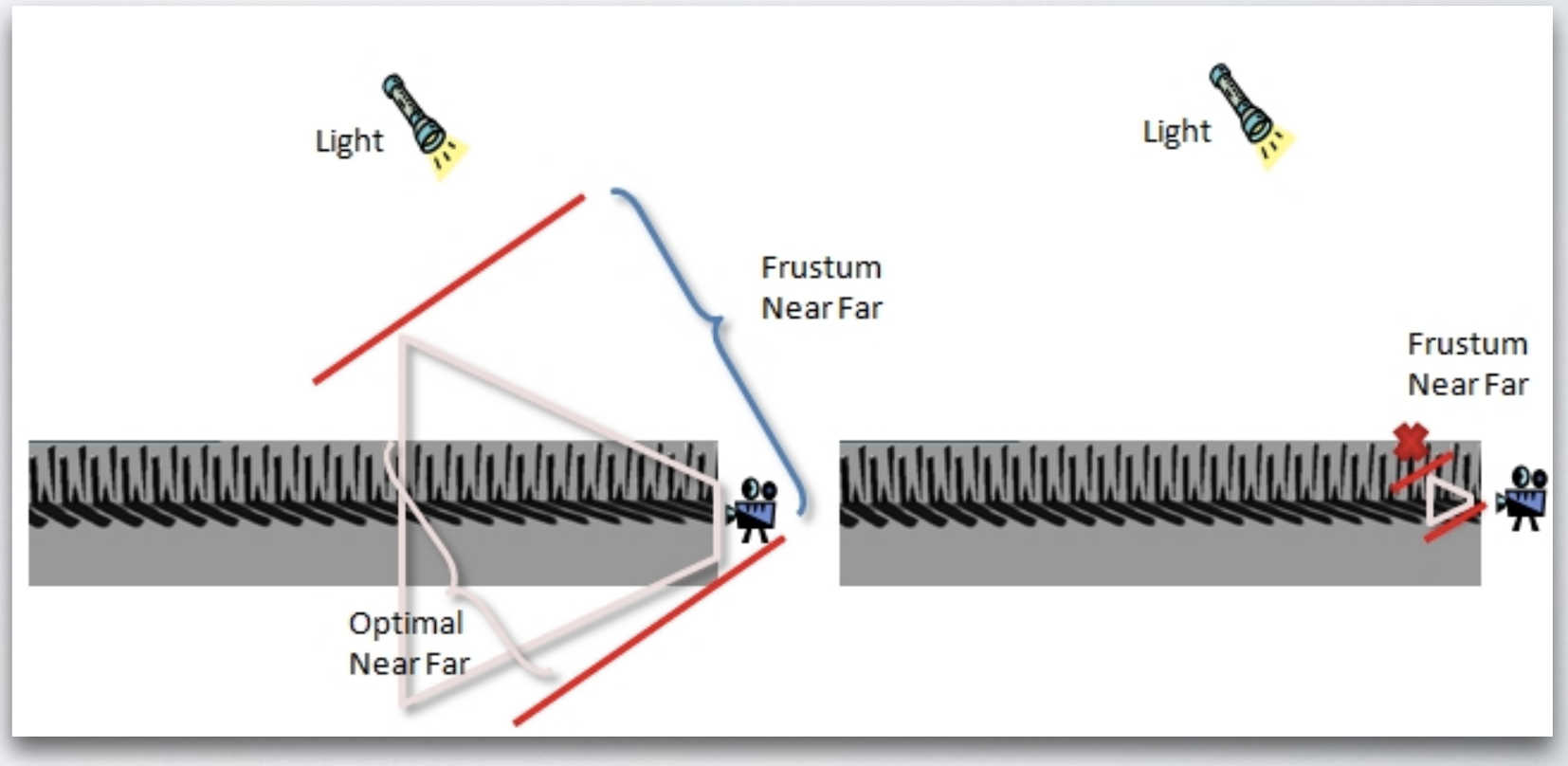
Calculating the near plane and far plane

- An easy and naive way to calculate the near plane and far plane is to transform the scene's bounding volume into light space.



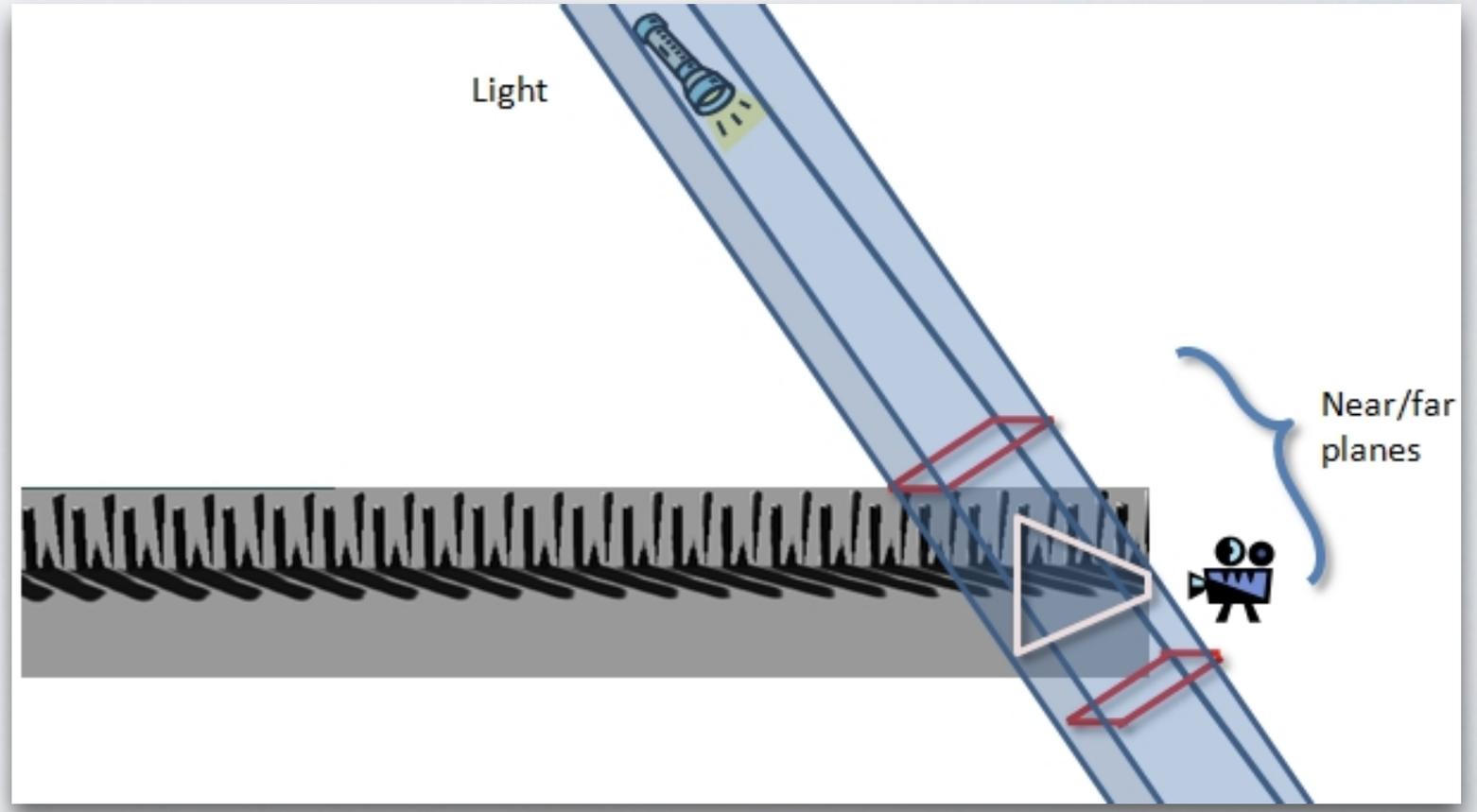
Calculating the near plane and far plane

- Another technique for calculating the near and far planes is to transform the frustum into light space and use the minimal and maximal values in Z as the near and the far planes, respectively.



Calculating the near plane and far plane

- The proper way to calculate the near and far planes is based on the light frustum intersection with the scene

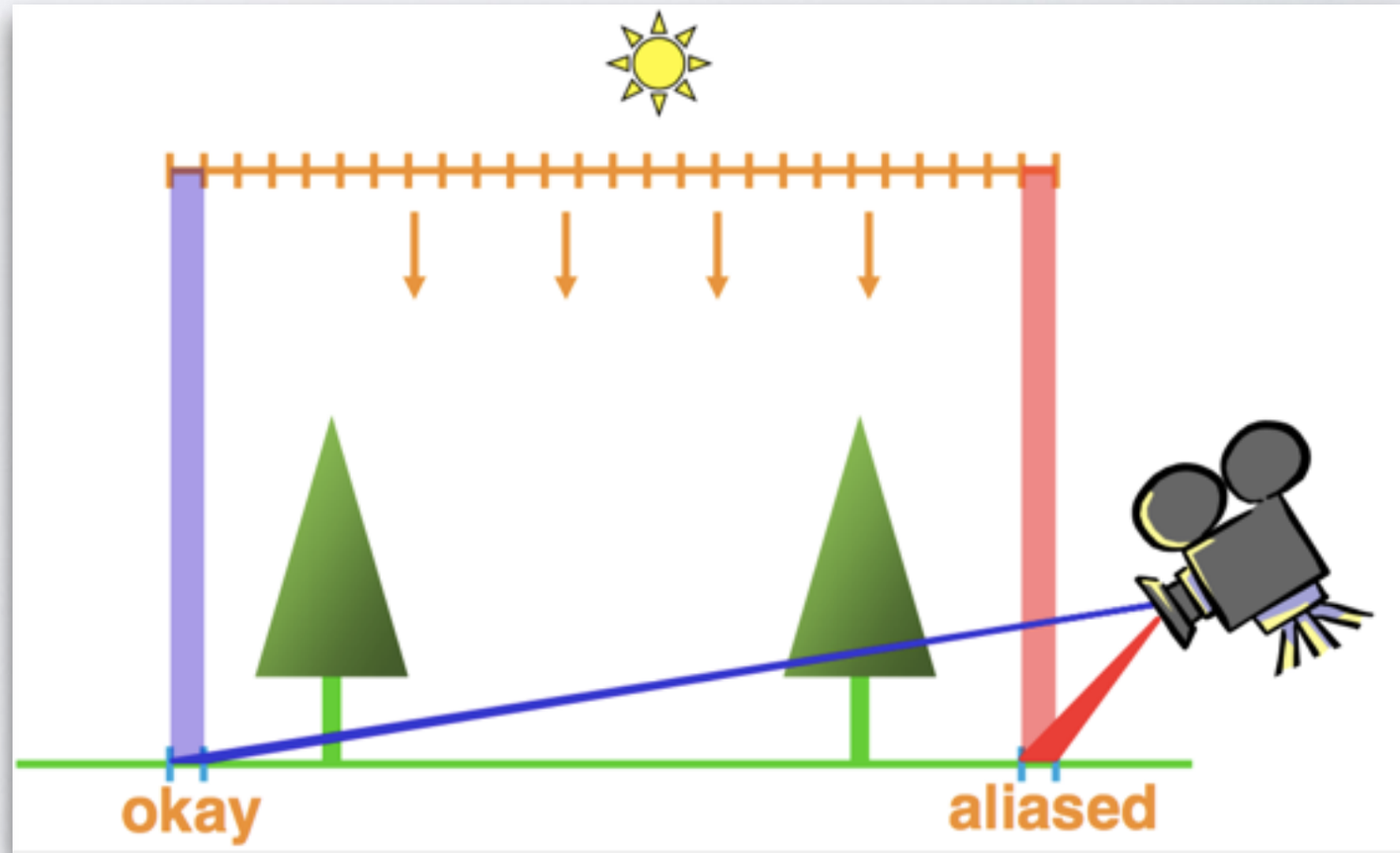


Fitting Calculation

- Model matrix **M**, light view matrix **Lv**, light projection matrix **Lp**;
- Transform intersection body by **Lp * Lv**
- Calculate bounds (xmin, ymin, xmax, ymax)
- Calculate fitting matrix **F**
- Shadow matrix: **S = F Lp Lv M**

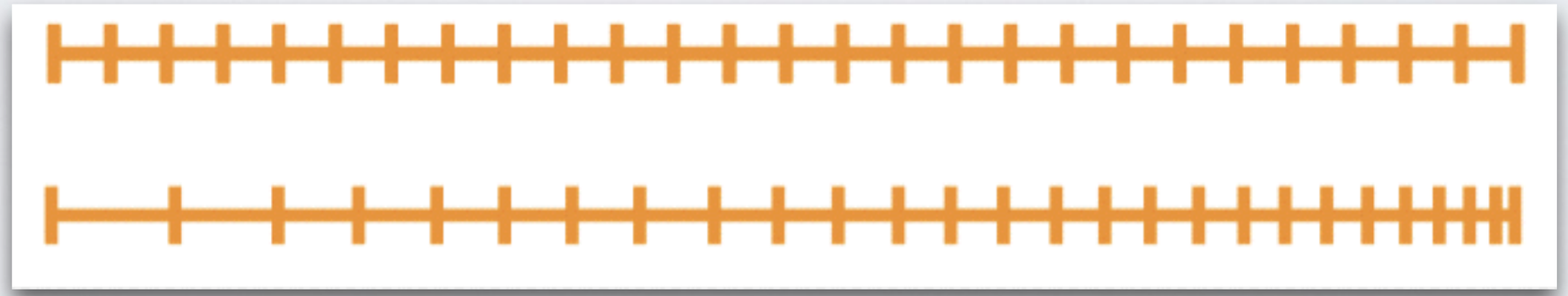
Wraping

- a solution for perspective aliasing
  - Insufficient resolution near eye



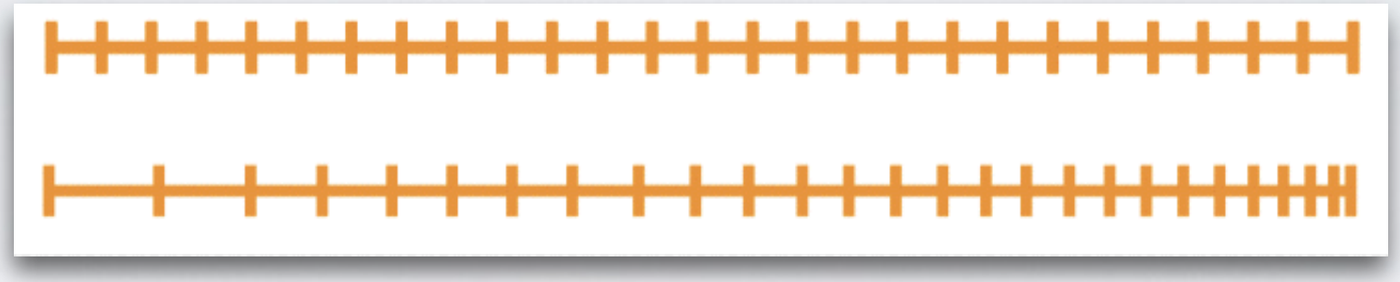
Solution for perspective aliasing

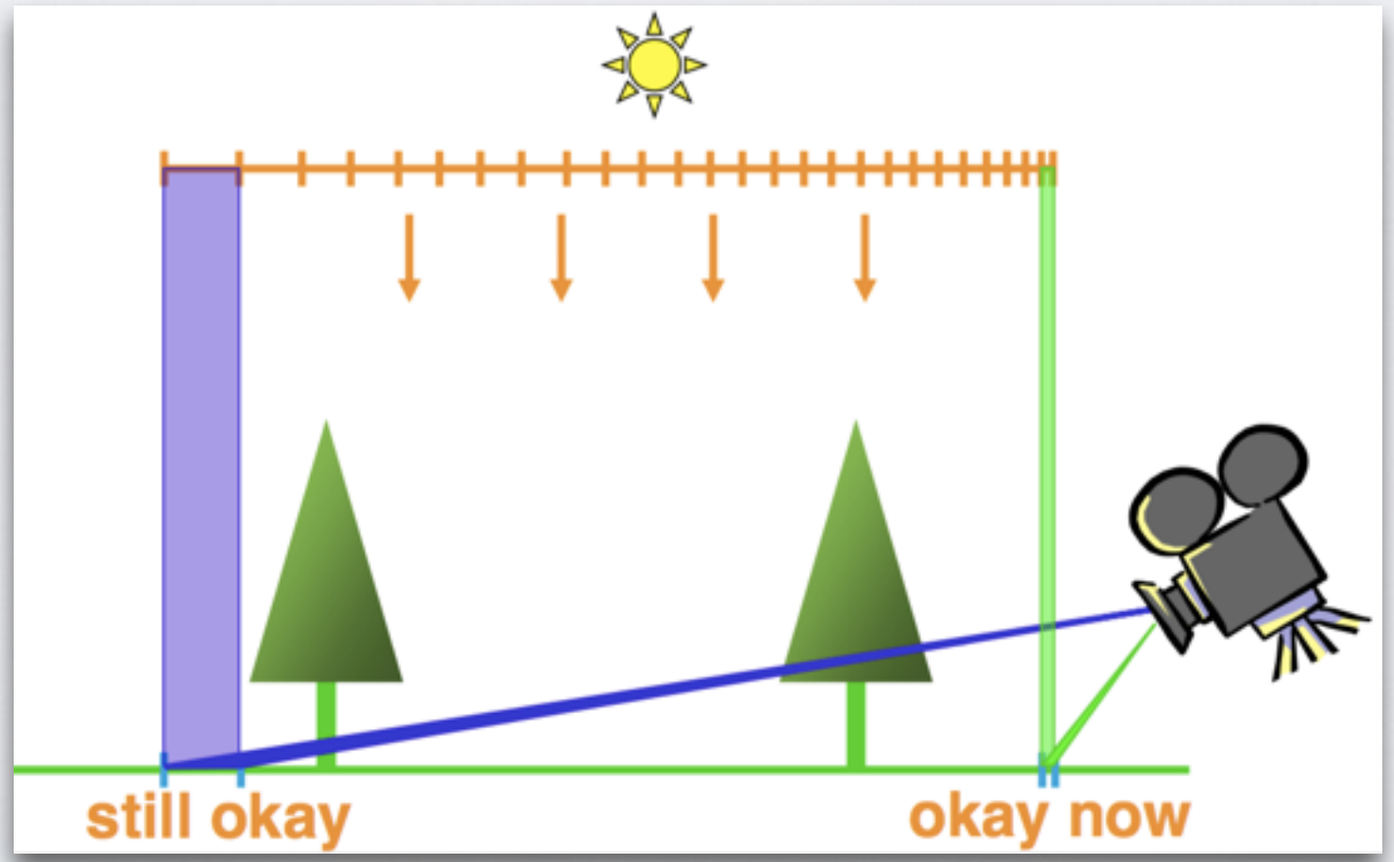
- redistribute values in shadow map



Solution for perspective aliasing

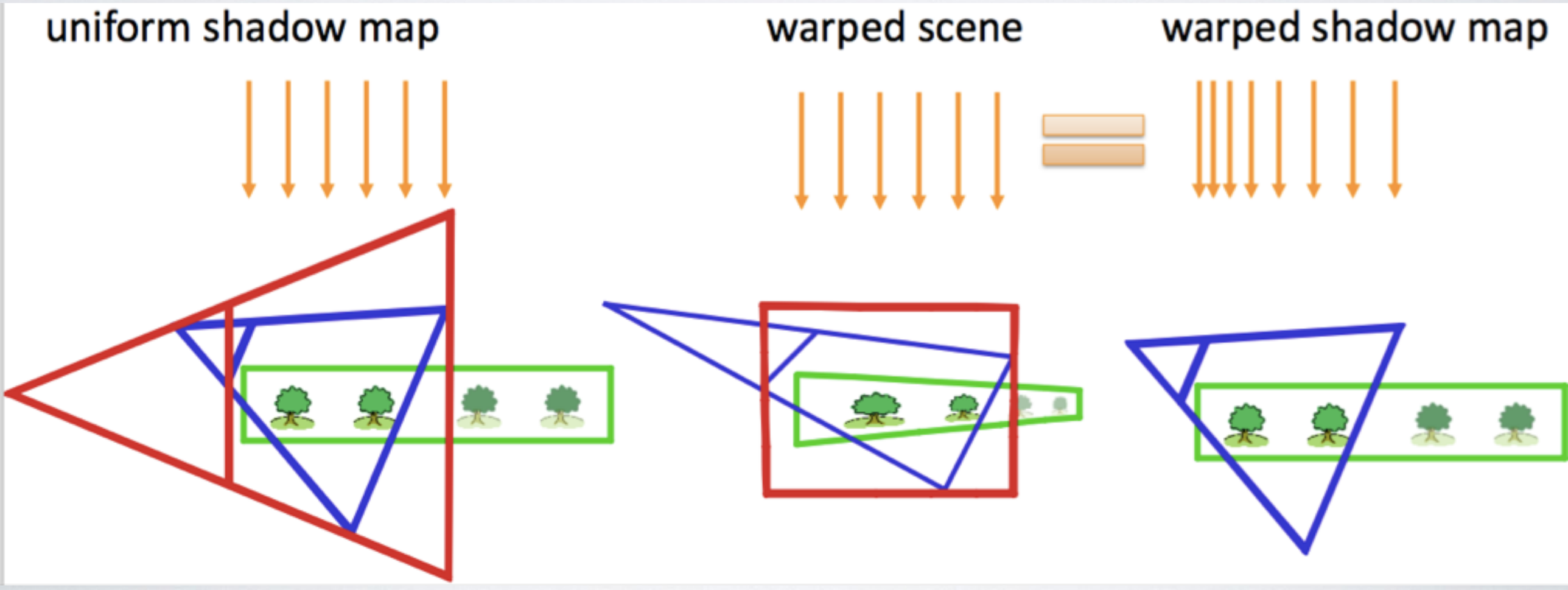
- redistribute values in shadow map





Shadow map wrapping

- use an additional perspective frustum
    - Perspective Shadow Map (PSM)



Shadow map wrapping

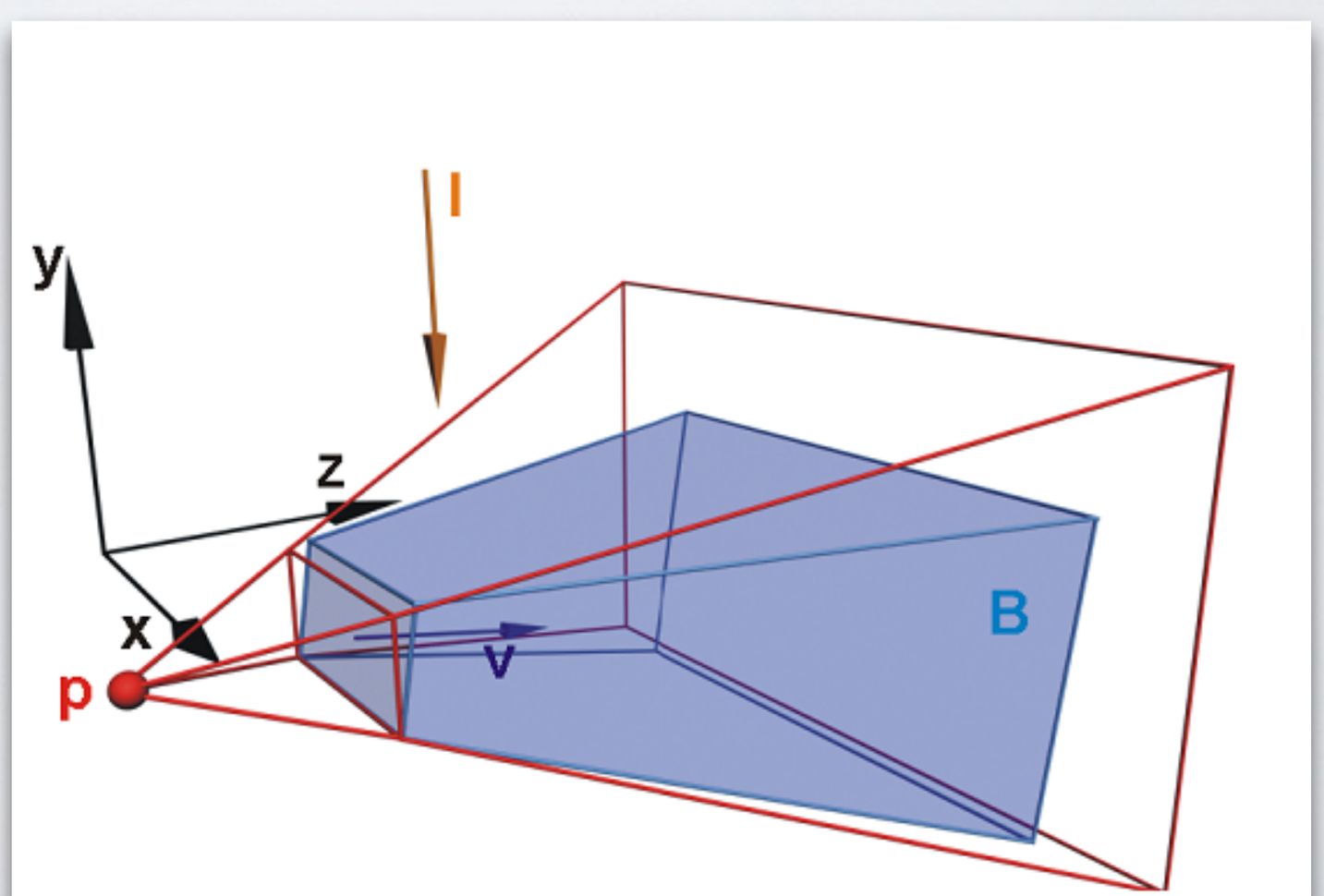
- PSM Problems:
  - Light change types
  - un-even z-distribution: very bad far away!

Shadow map wrapping

- Enter Light-Space Perspective Shadow Map (LiSPSM)
    - Perspective frustum defined relative to light space;
    - Optimize wrapping strength to improve z-distribution

Light-Space Perspective Shadow Map

- Light and view vector define yz-plane
- Find a tight perspective frustum on focused region
- In light space



LiSPSM Matrix

1. Standard light matrix: **S = F Lp Lv M**
2. Orient shadow map along view vector: **R**
3. Transform intersection body by: **R Lp Lv**
4. Find near/far, and choose warping strength: **n**
5. Calculate warping matrix: **Wp Wv**
6. Calculate **F** using: **Wp Wv R Lp Lv**
7. Wraped shadow matrix: **Sw = F Wp Wv R Lp Lv M**
8. Use **Sw** both for shadow-map generation and rendering

Partitioning

- One of the best way to combat perspective aliasing
- Partition view frustum into n sub-frusta
- Calculate separate shadow map for each
- Z-Partitioning
    - Parallel Split Shadow Maps (PSSM)
    - Cascaded Shadow Maps (CSM)

Cascaded Shadow Maps

- The basic idea of CSMs is to partition the frustum into multiple frusta. A shadow map is rendered for each subfrustum; the pixel shader then samples from the map that most closely matches the required resolution

CSM Algorithm

1. Partition the frustum into subfrustum.

2. Compute an orthographic projection for each subfrustum.

3. Render a shadow map for each subfrustum.

4. Render the scene

    1. Bind the shadow maps and render.

    2. The vertex shader does the following:

        - Compute texture coordinates for each light frustum.

        - Transforms and lights the vervex.

    3. The pixel shader does the following:

        - Determines the proper shadow map.

        - Transform the texture coordinates if necessary.

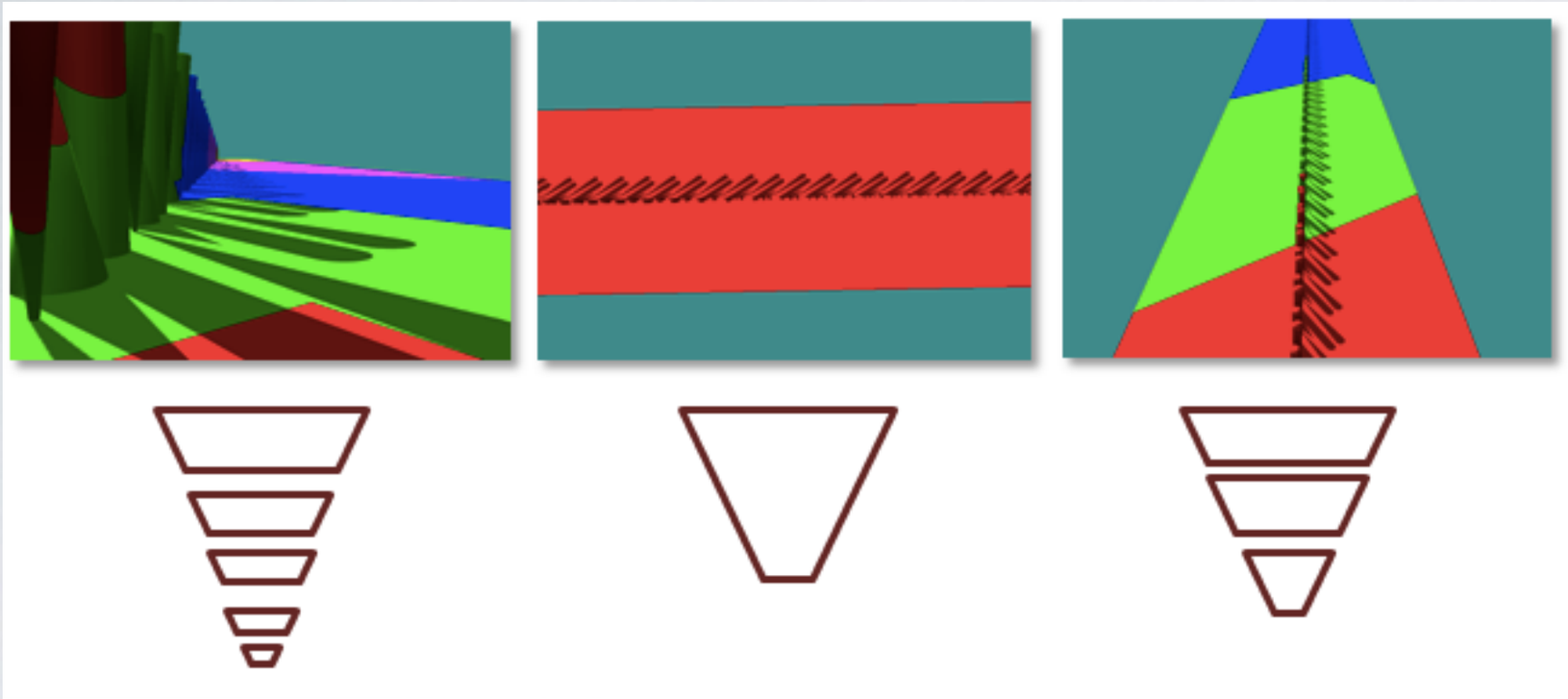
        - Sample the cascade

        - Lights the pixel

Partitioning the Frustum

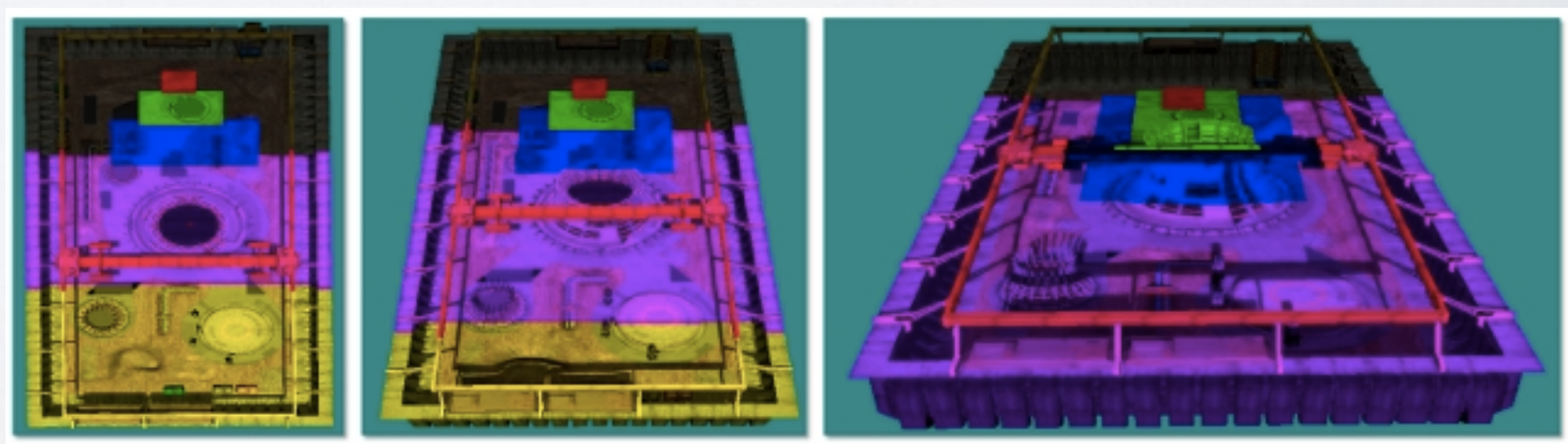
- Partitioning the frustum is the act of creating subfrustum;
- Factors should taken into account:
    - Orientation of the scene geometry
    - Orientation of the light and camera

Partitioning the Frustum



Different configurations require different frustum splits

Cascade overlap increases as light direction becomes parallel with camera direction

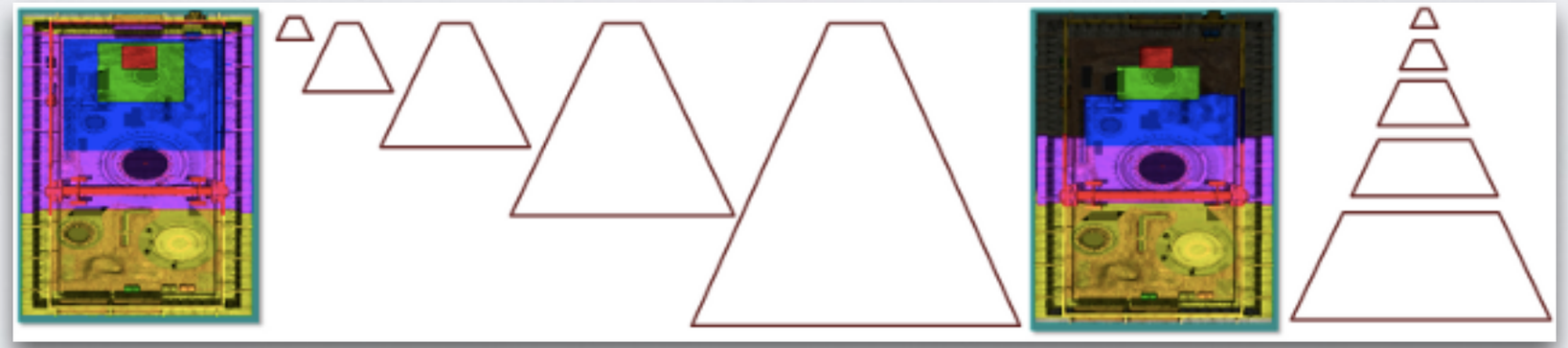


Calculating a View-Frustum Bound

- Once the frustum intervals are selected, the subfrusta are created using one of two:
  - Fit to scene
  - Fit to cascade

Calculating a View-Frustum Bound

Fit to scene vs. fit to cascade



Render the Shadow Map

- using *texture array* in today's GPU;
- using different light viewprojection matrix for each cascade.

Render the Scene

- Two methods for selecting the cascaded:
  - Interval-Based Cascaded Selection
  - Map-Based Cascaded Selection

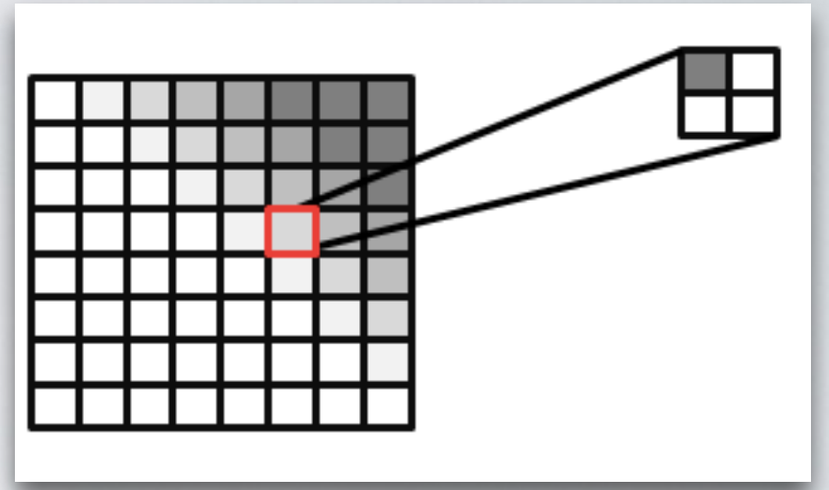
Blend between Cascades

- a visible seam between cascade layers because the resolution does not match;
- solution: expand each cascade slightly to create an overlap, then blend between two cascades at the boundaries.

Improve Resampling

- Shadow Maps Filtering

Percentage Closer Filter



- serves to move the hard edge;

- calculate a percentage of the pixel in shadow based on the number of subsamples that pass the depth test over the total number of subsamples;

- support by hardware;

- can extended to larger kernels without hardware support.

Depth Bias in PCF

- Depth bias becomes even more important when large PCF kernels are used;
- A larger bias offset can alleviate the erroneous self-shadowing, but too large of an offset can lead to Peter Panning.

Variance Shadow Maps

- Hardware PCF does't support filtering;
- Idea : comparing a single value to a particular distribution of values(the region we're sampling), and calculate the percentage of those values are greater than the single value.
- using *Chebyshev's inequlity*

$$P_{(x \geq t)} \leq \frac{\sigma^2}{\sigma^2 + (t - E(x))}$$

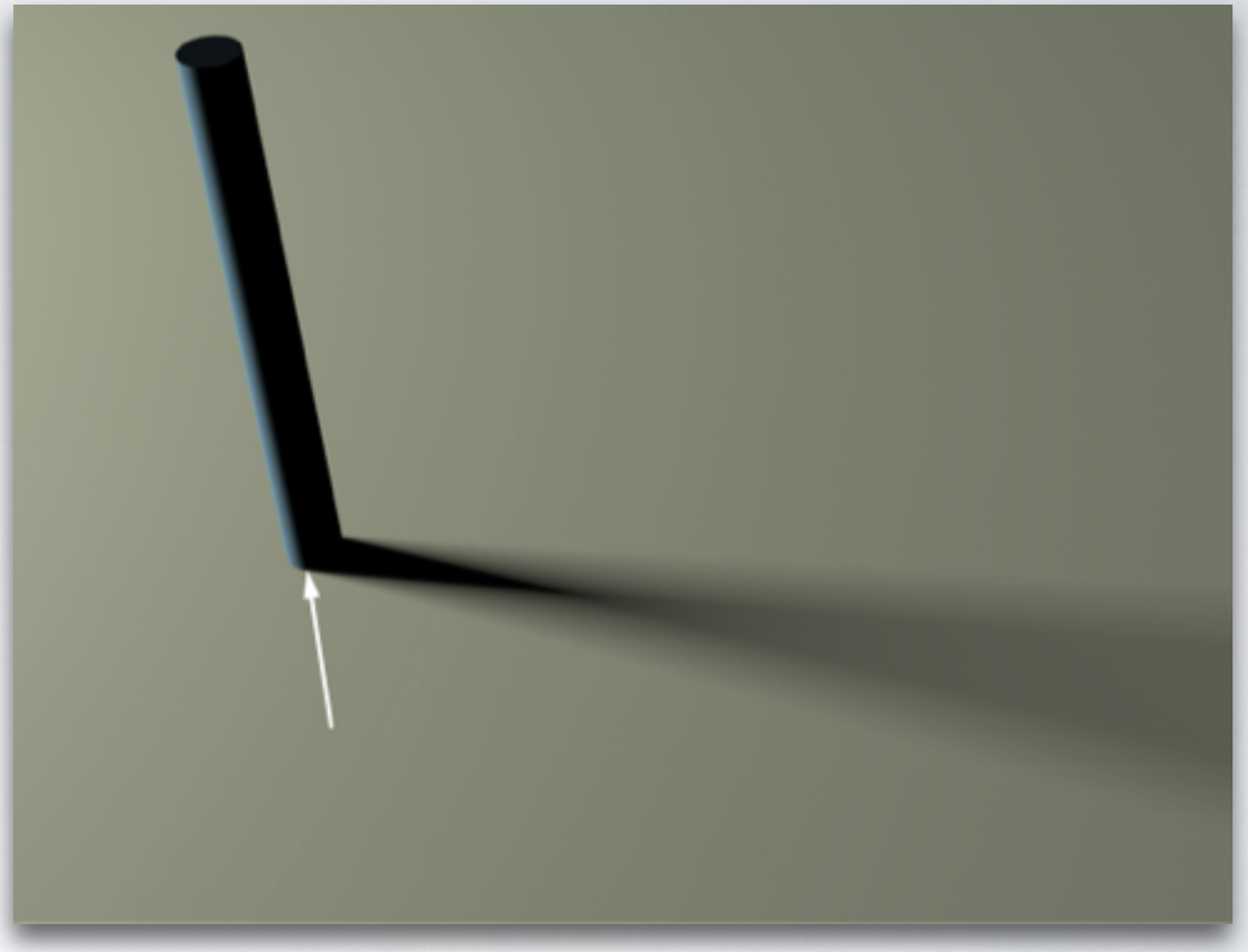
VSM Algorithm

1. Render shadow map, store depth, depth^2;

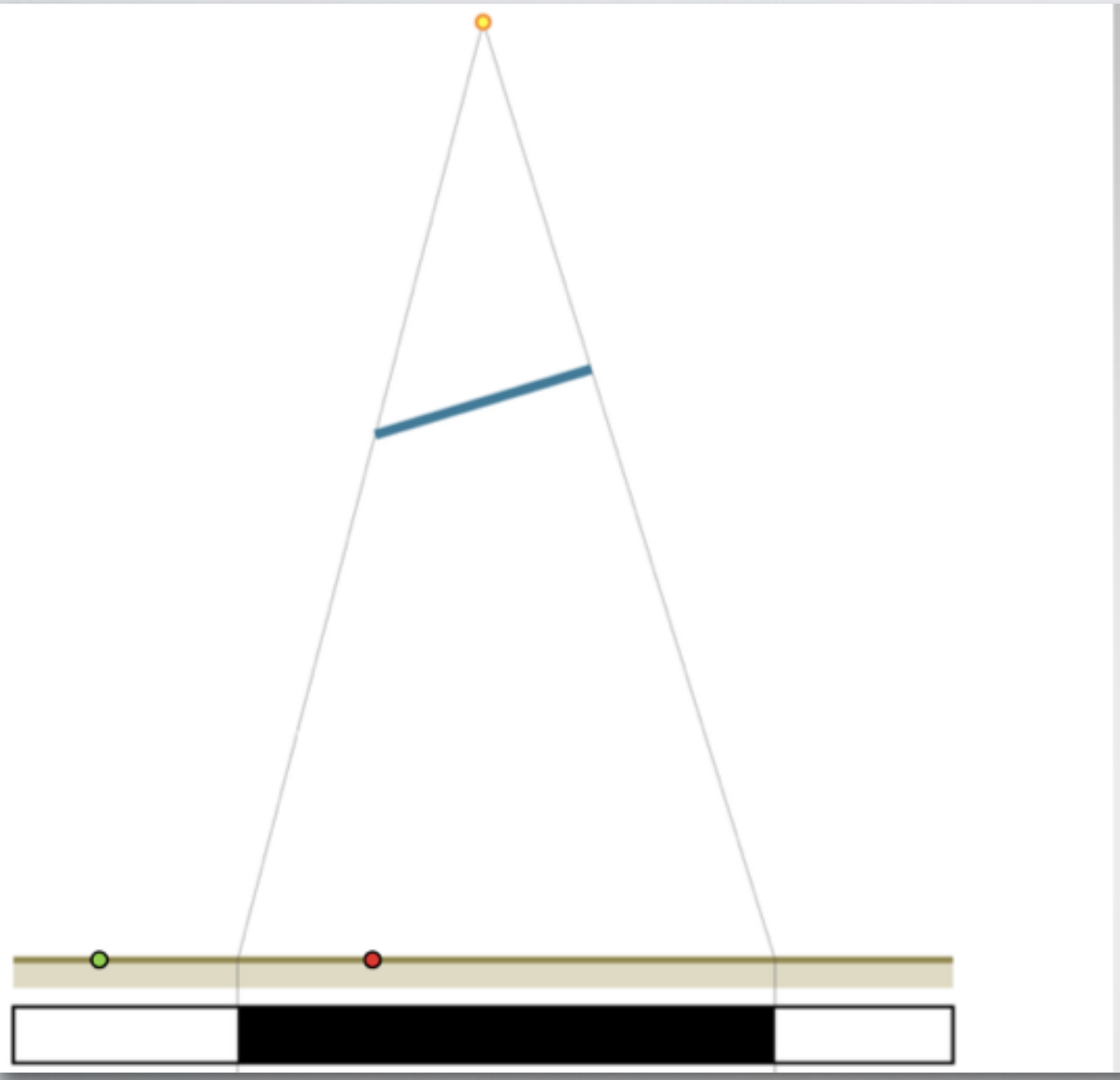
2. Filter the shadow map;

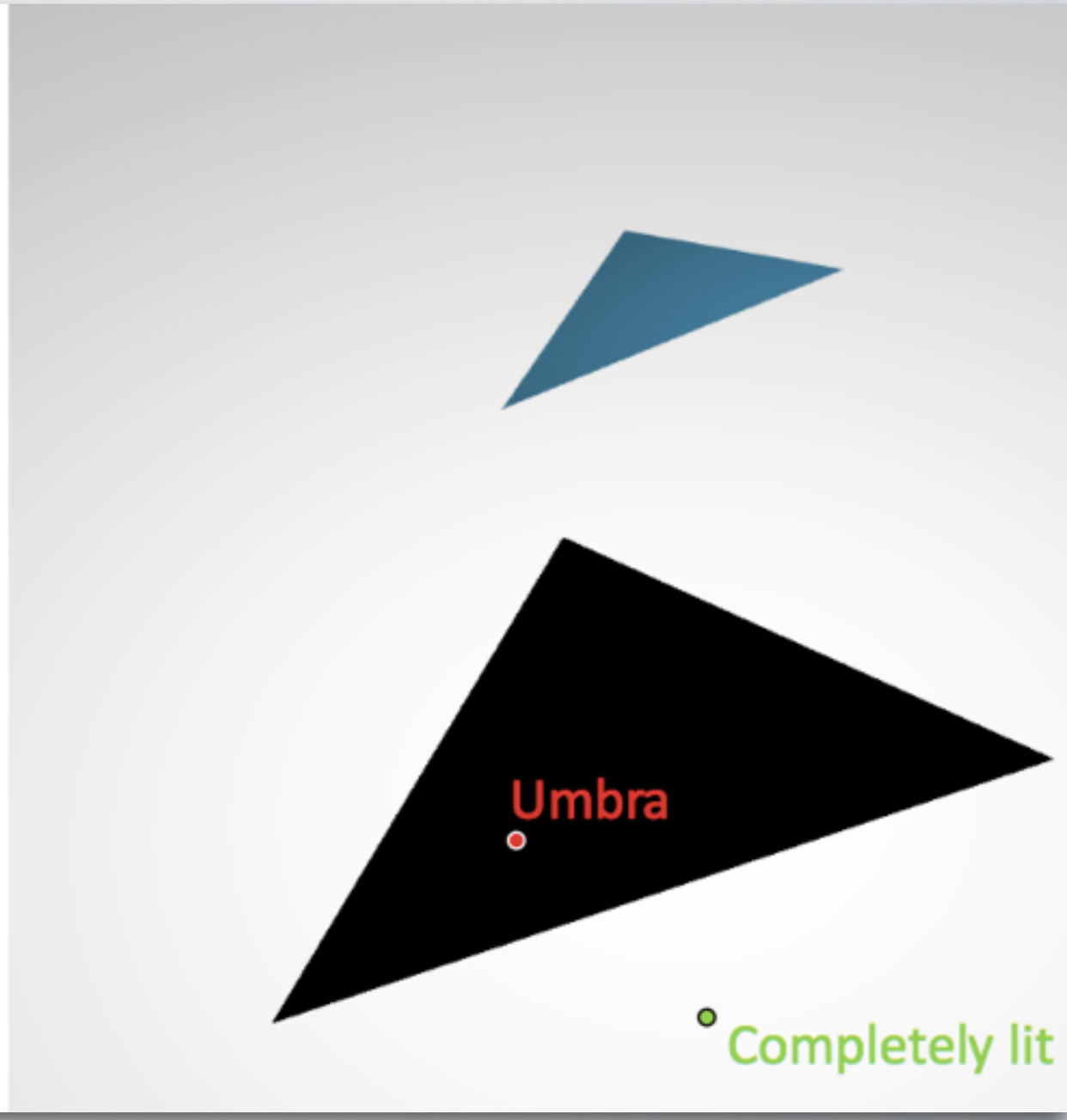
3. Render scene with shadow using *Chebyshev's inequlity.*

SOFT SHADOWS

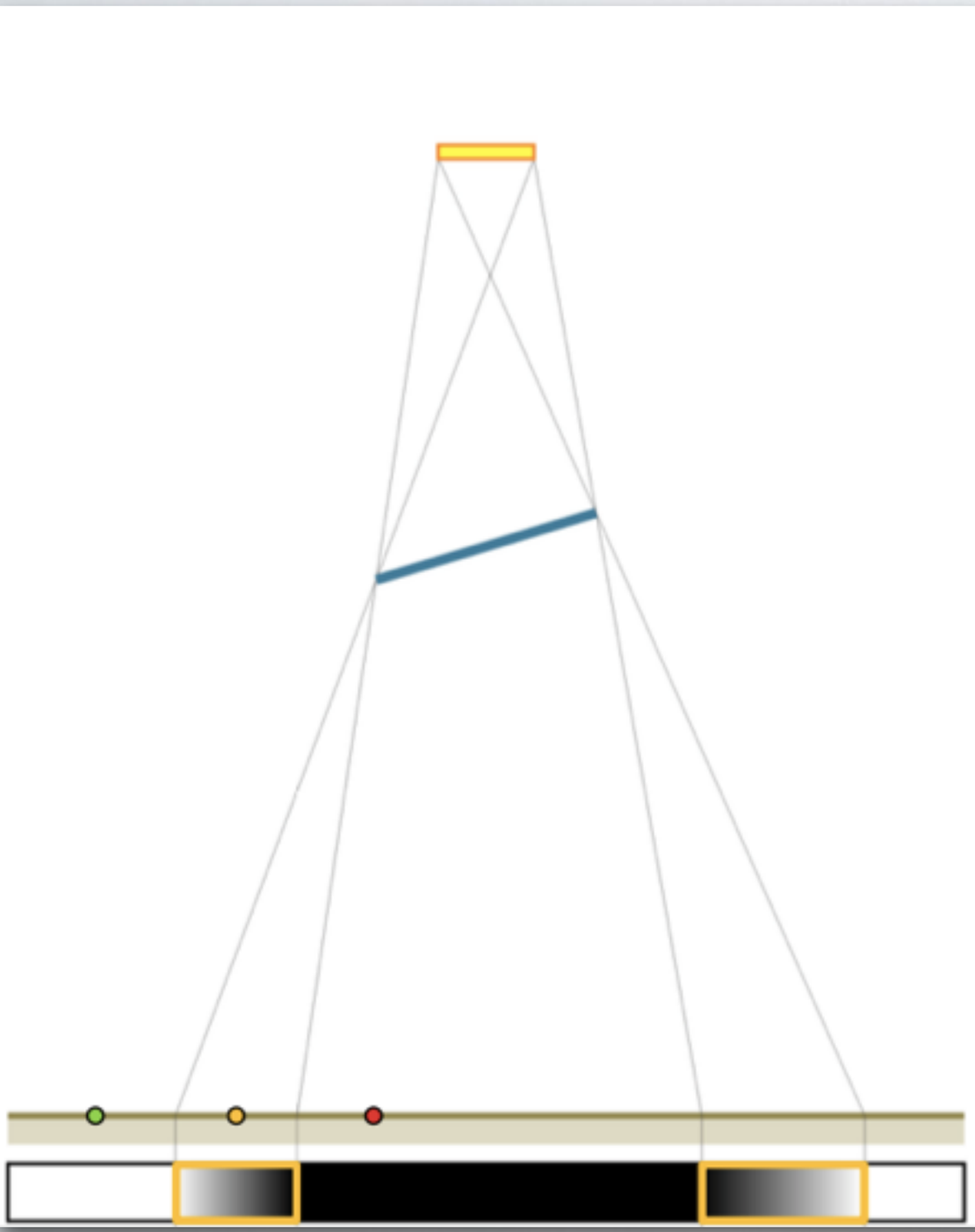


Hard Shadows





Soft Shadows



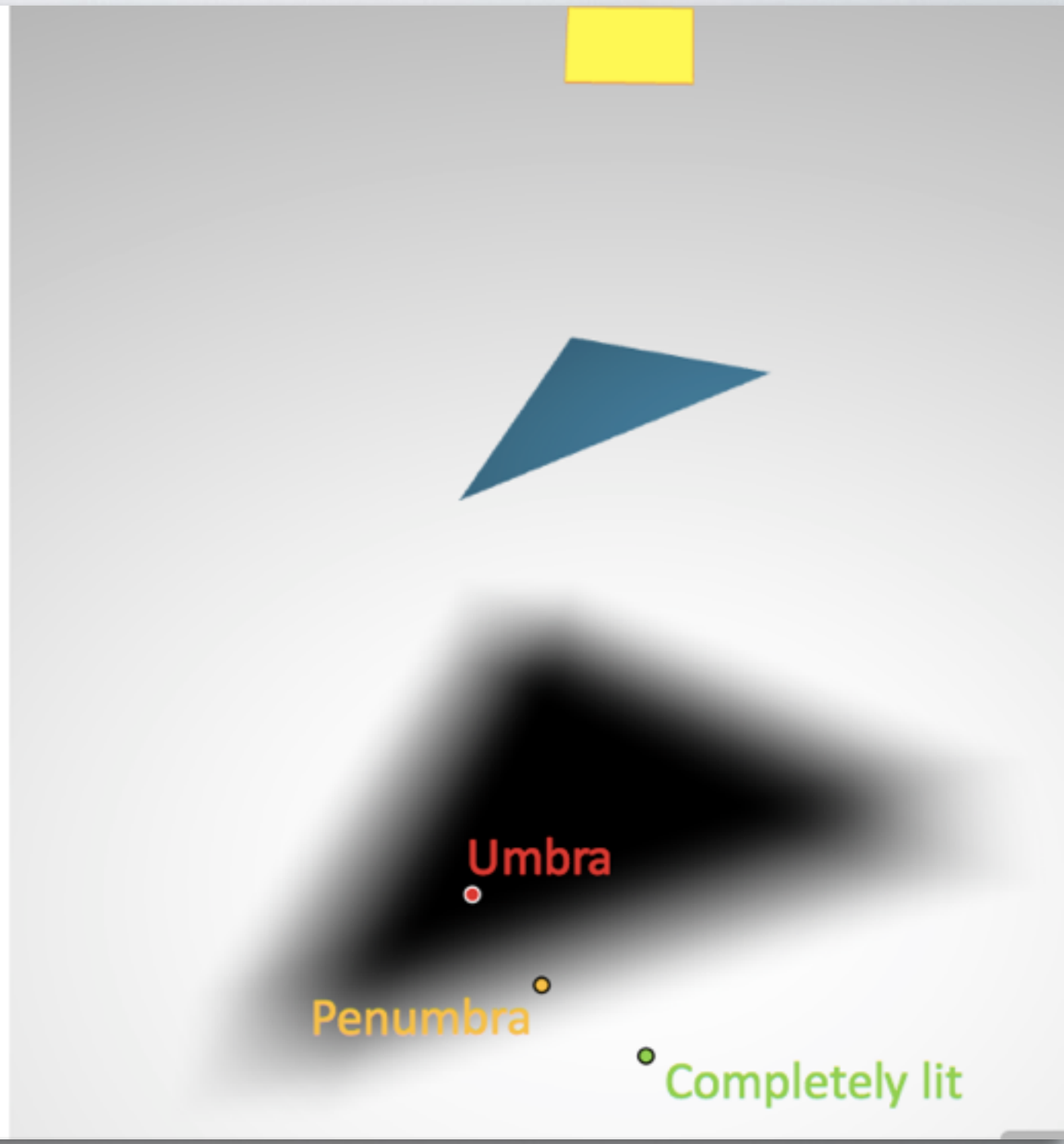


Image-based solutions

- Use any of the filtering approaches presented before yields a soft-shadow-like appearance
- But: ignores varying penumbra width.

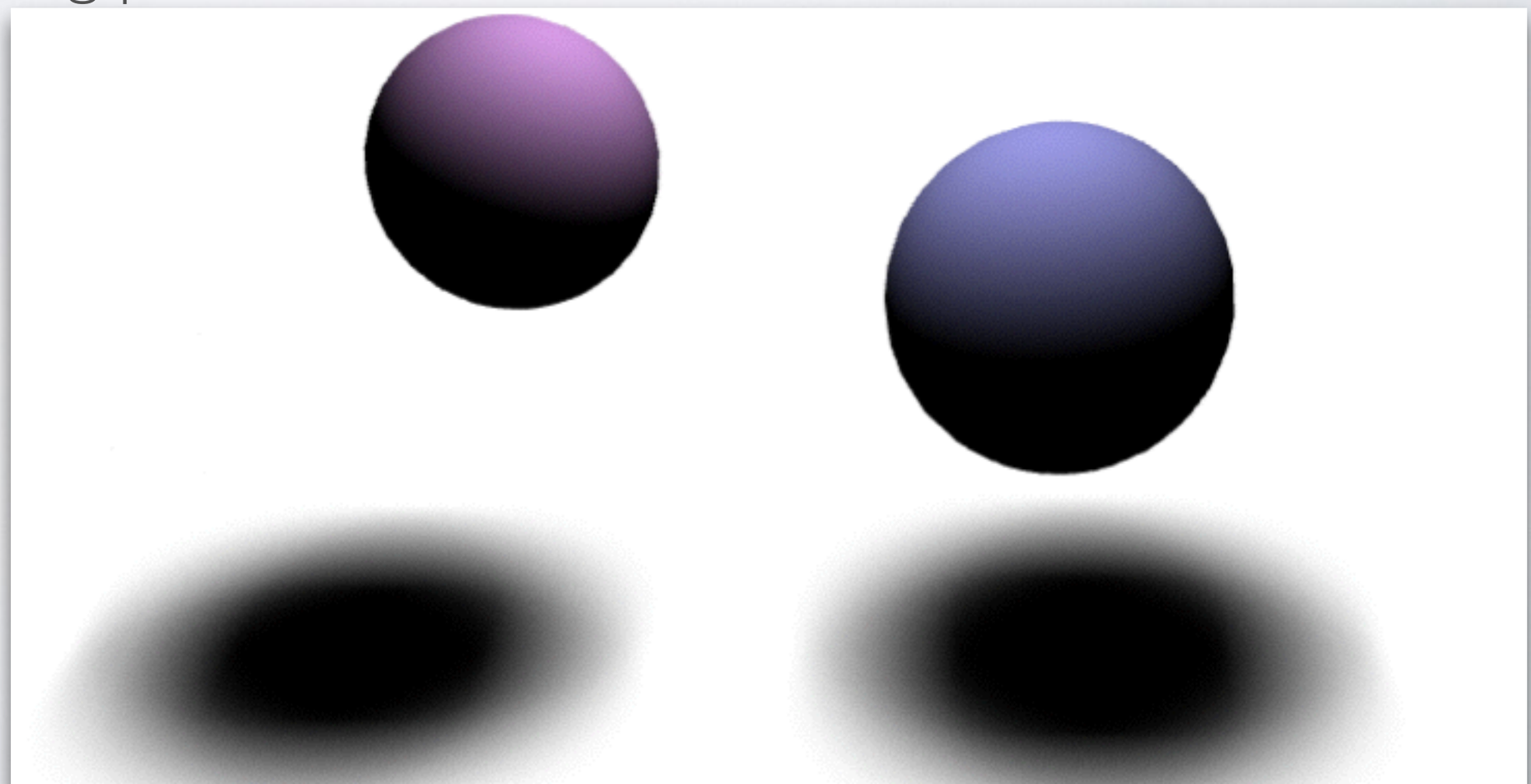
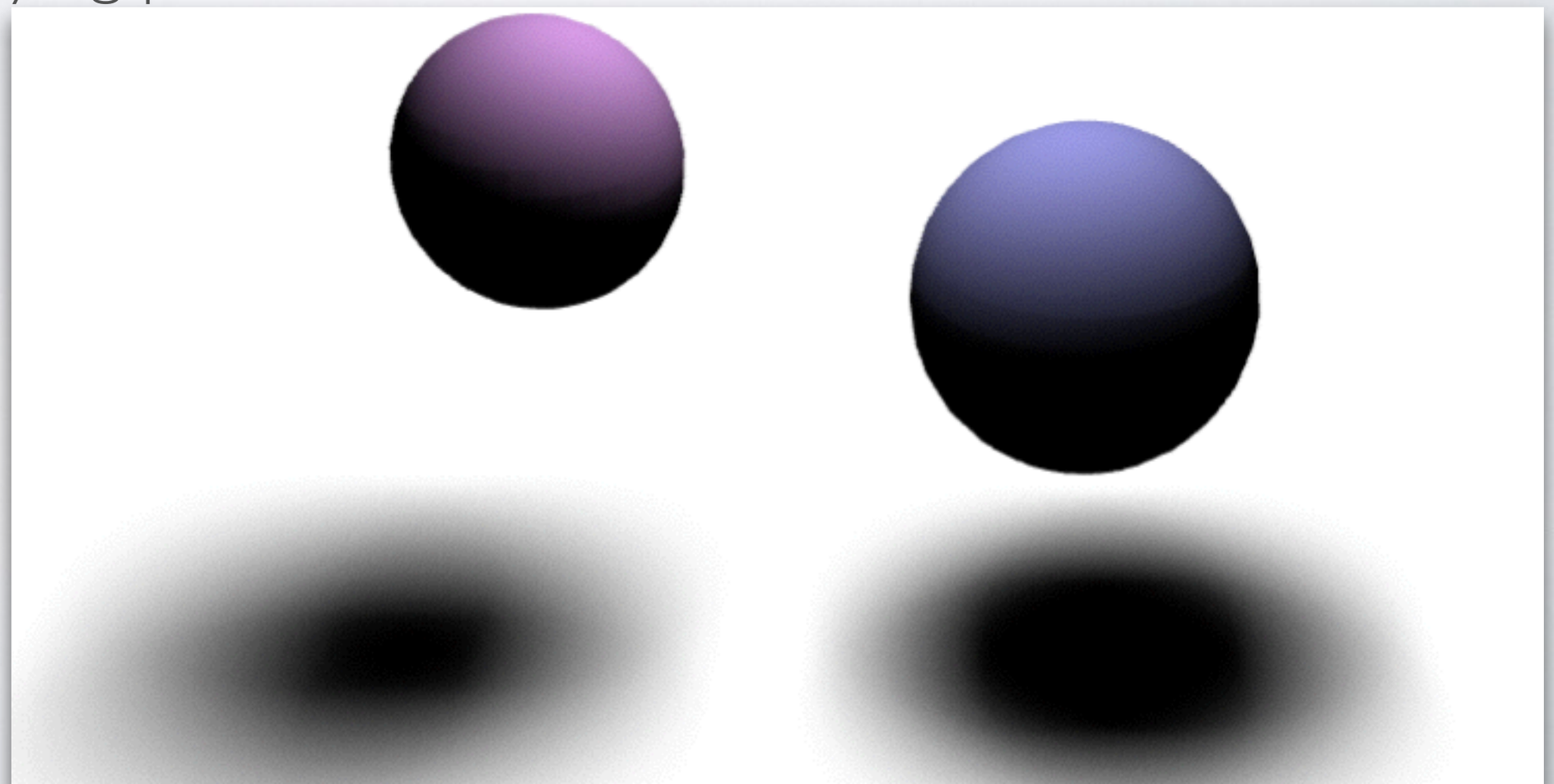


Image-based solutions

- Use any of the filtering approaches presented before yields a soft-shadow-like appearance
- But: ignores varying penumbra width.

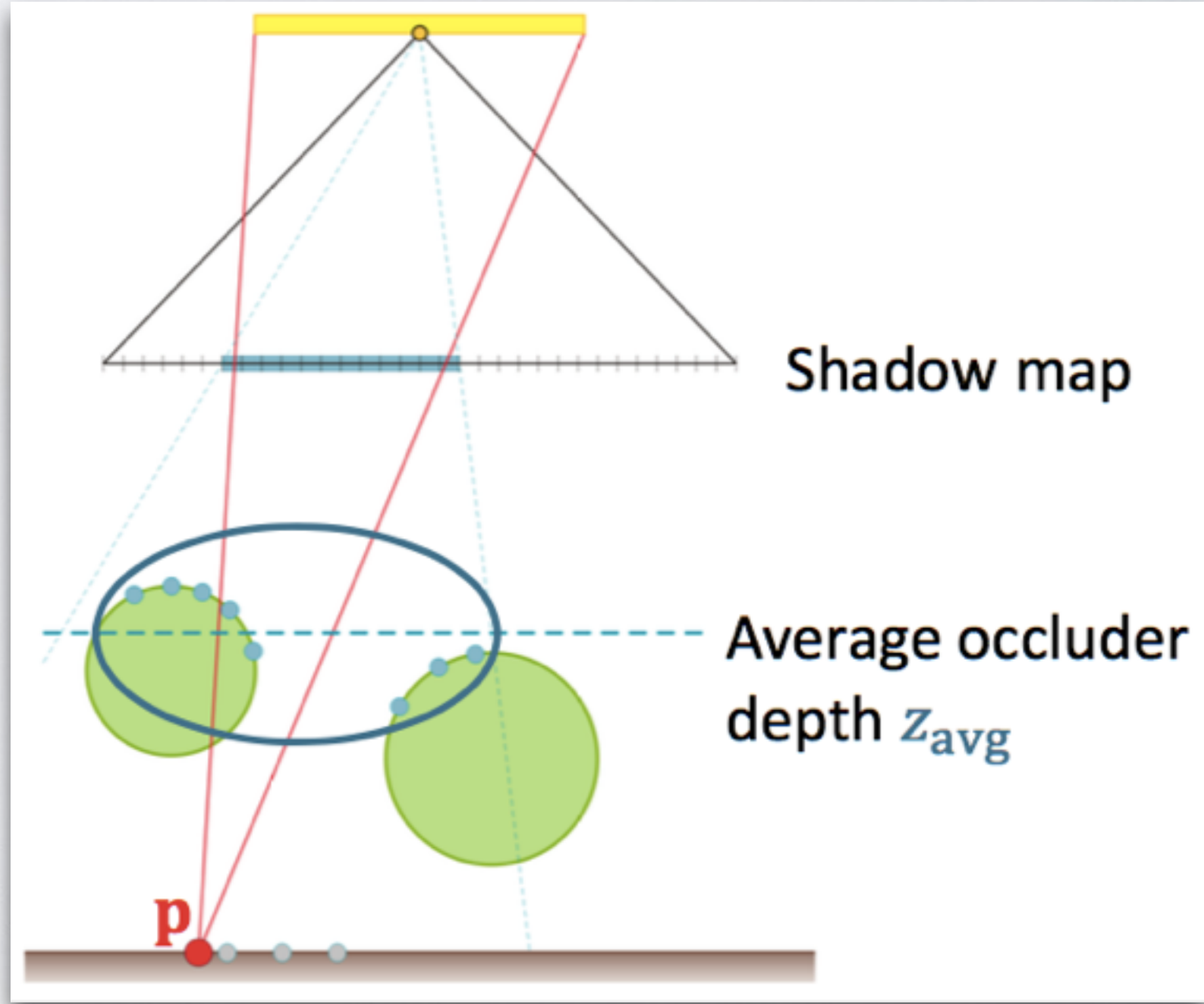


Percentage-Closer Soft Shadow(PCSS)

- idea: choose blur kernel size adaptively

Percentage-Closer Soft Shadow(PCSS)

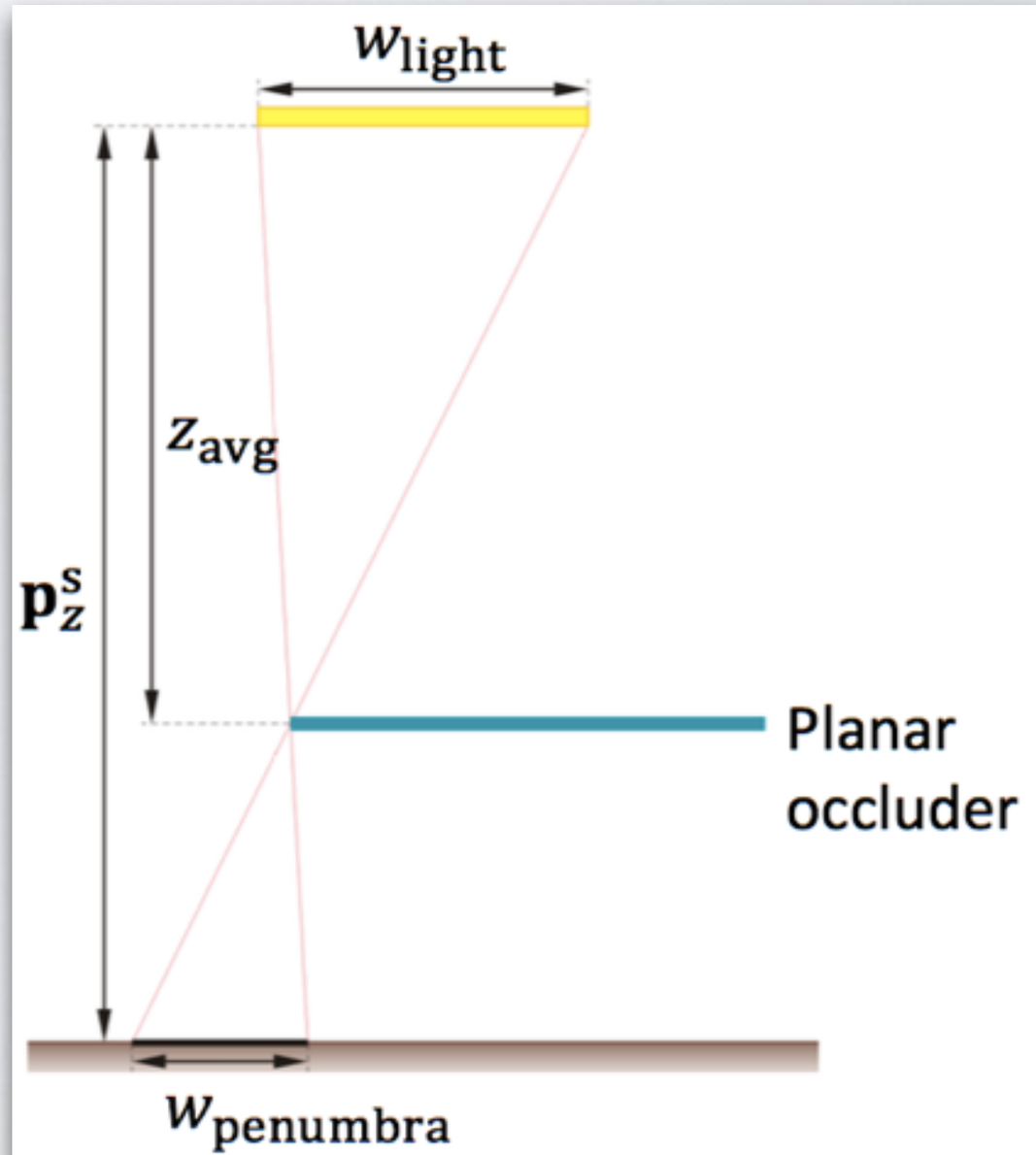
- 1.) Blocker search, early out if there is none



Percentage-Closer Soft Shadow(PCSS)

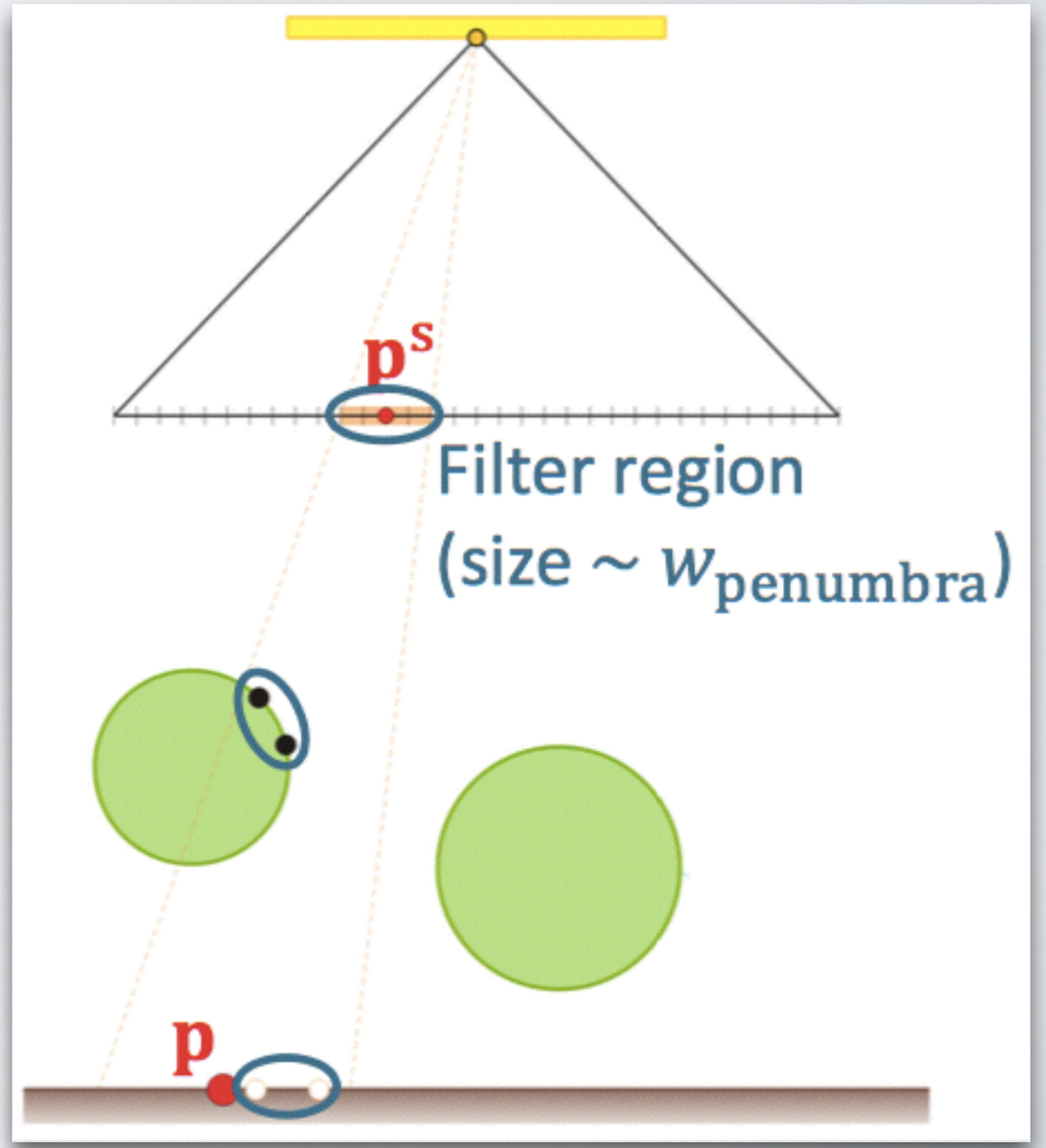
- 2.) estimate penumbra width

$$w_{penumbra} = \frac{P_z^s - z_{avg}}{z_{avg}} w_{light}$$



Percentage-Closer Soft Shadow(PCSS)

- 3.) apply filtering (PCF)



Assignment

- Modify the shadow demo, change the standard shadow map projection from perspective to orthographic;

- Or make your own shadow map implementation.