Computer Graphics 2

Shadows

Marek Zimányi

Michal Valient

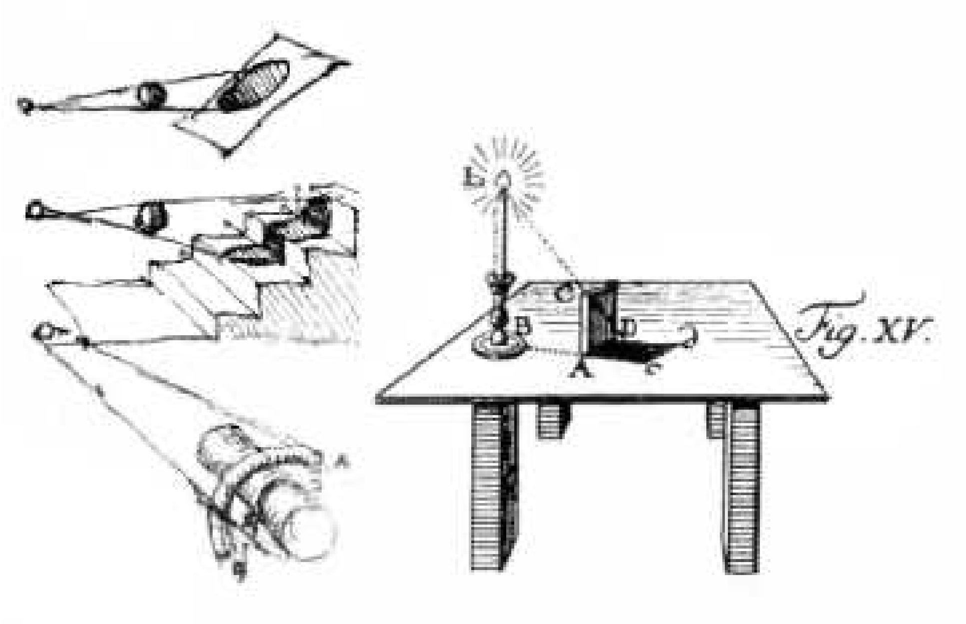
Katedra aplikovanej informatiky,

OPGSO, FMFI UK, Bratislava

2004/2005

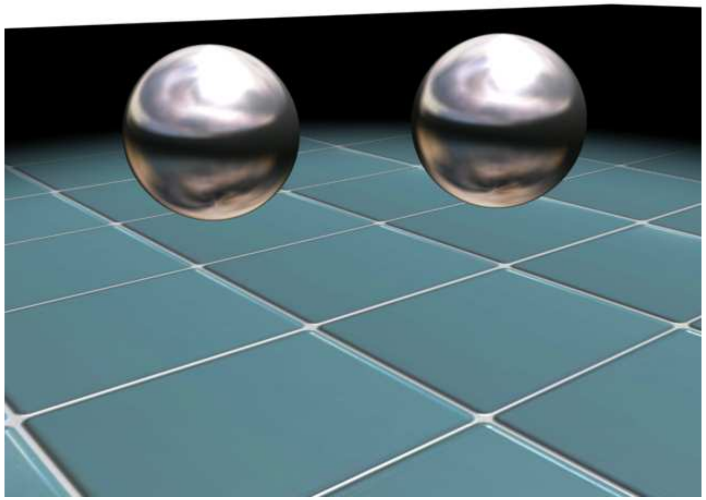
History

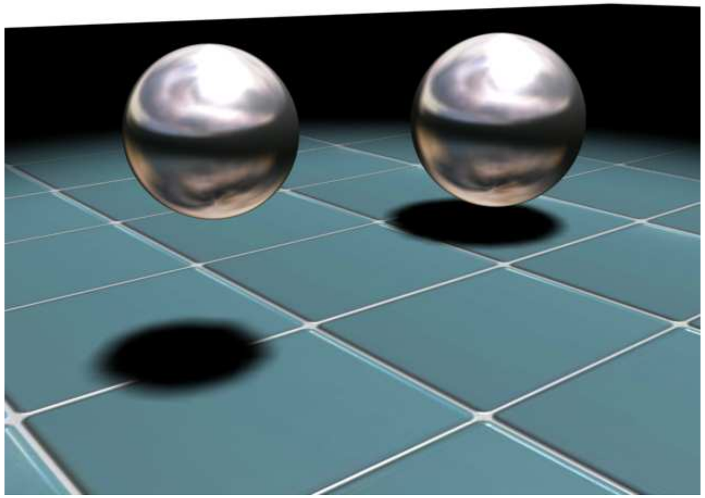
- Leonardo Da Vinci



Importance of shadows

- Perception
  - Element of the orientation
  - Relationship between objects
- Artistic element
  - Expressing the mood of the scene
  - Hard/soft shadows





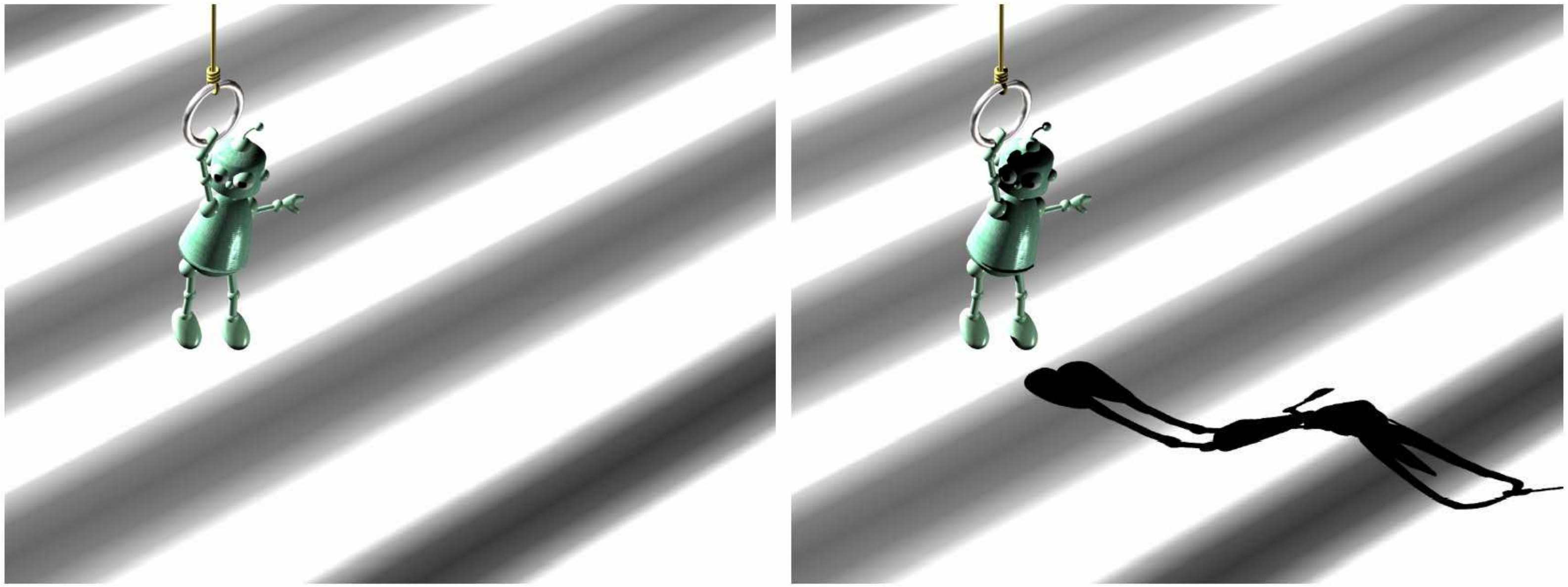
Importance of shadows II

- Shadows provide extra information

Hasenfratz et al - **A Survey of Real-time Soft Shadows Algorithms**

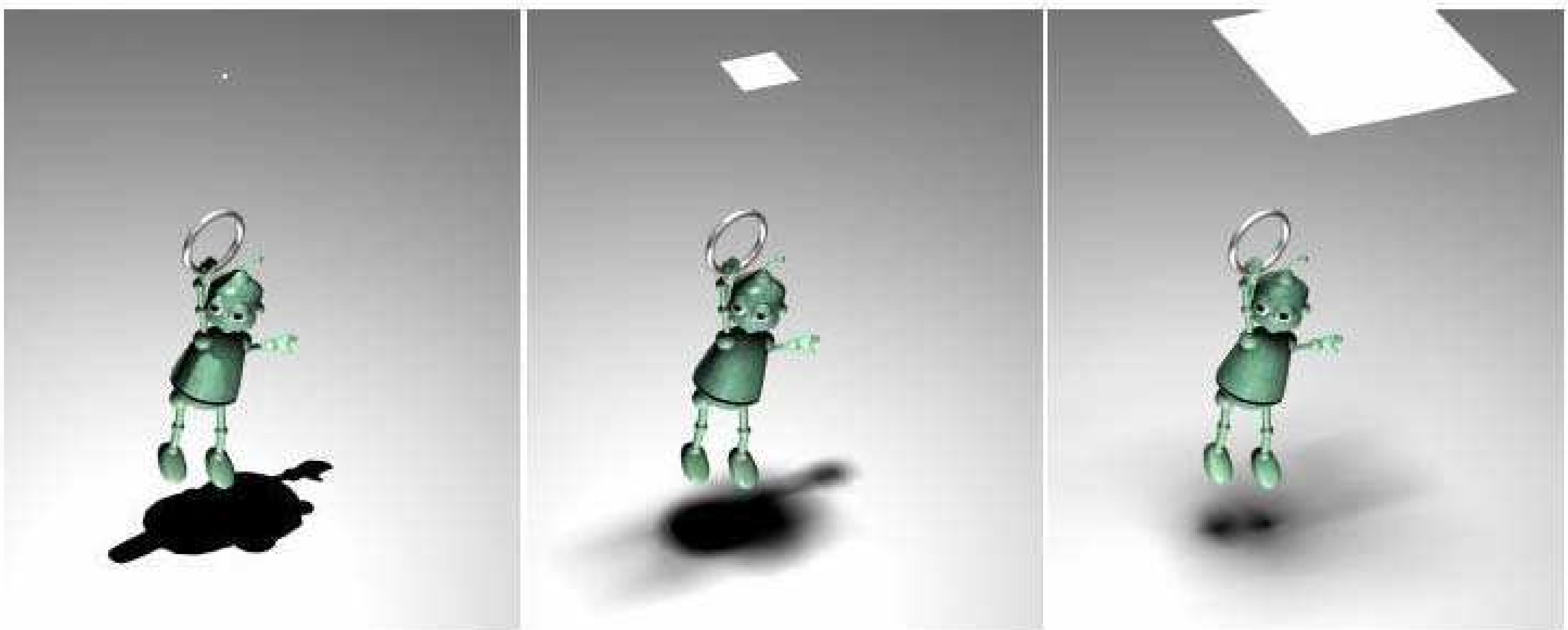
Importance of shadows II

- Shadows provide information about the geometry of the receiver

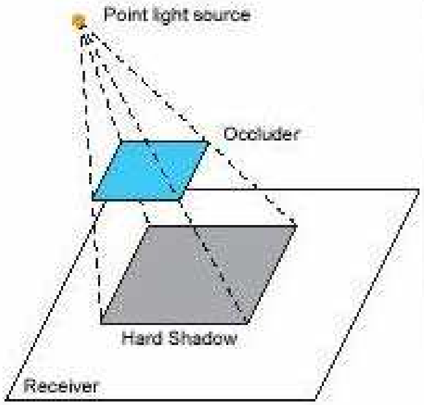


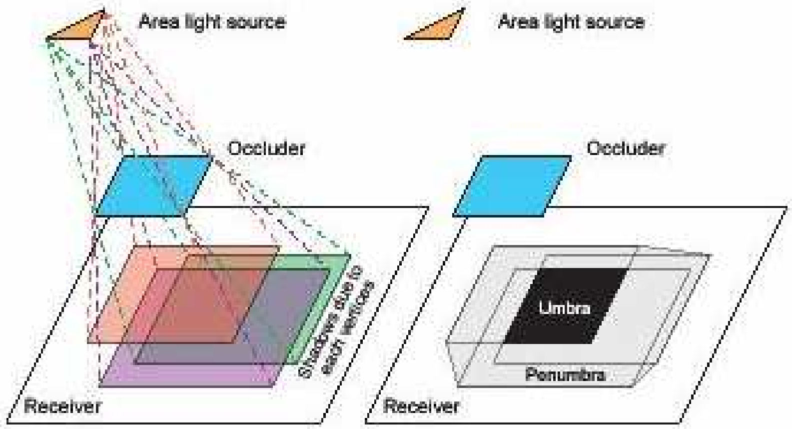
Hasenfratz et al - **A Survey of Real-time Soft Shadows Algorithms**

Area lights



Hard vs soft shadows

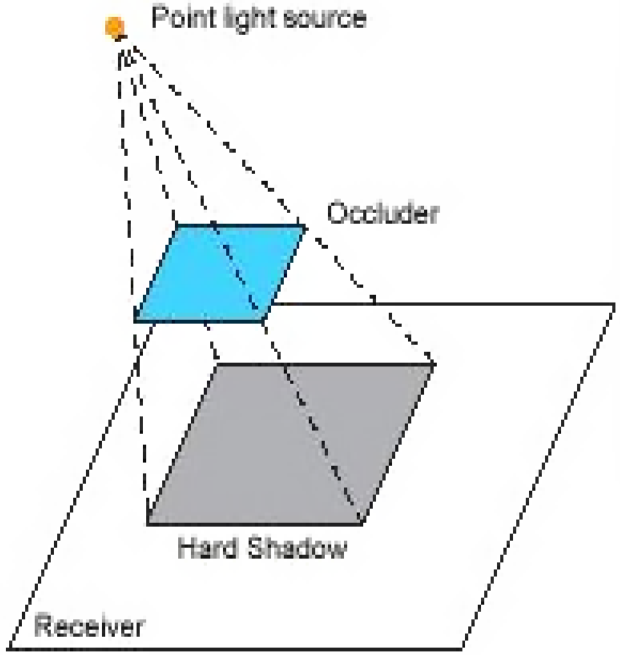




Multiple light sources

Terms

- Light sources
  - Point, area, cone, hemi, ambient
- Occluders & receivers
  - Occluder - shadow volume
  - Receiver (object with shadow)
- Shadows
  - Umbra (source completely occluded)
  - Penumbra (source partially occluded)



Terms II

- Shadow sharpness
  - Soft shadows
  - Hard shadows

- Shadow type
  - Self shadow
  - Cast shadow

How to add shadows into scene?

$$I = I_a k_a O_{d\lambda} + \sum_{1 \le i \le m} S_i f_{att_i} I_{L_i} \left(k_d O_{d\lambda} \left(\vec{L} \cdot \vec{N} \right) + k_s O_{s\lambda} \left(\vec{R_i} \cdot \vec{V} \right)^n \right)$$

$$S_i = \begin{cases} 0, \text{ if light source } i \text{ is not visible from the calculated point} \\ 1, \text{ other} \end{cases}$$

- Projective shadows
- Shadow volumes
- Shadow maps
- Shadow optimizations

Shadows in OpenGL

- Fake shadows
  - Simple improvement of the spatial impression
- Analytical methods
  - Calculation of the projective transformations
- Rasterization methods
  - Algorithm works on the raster in the scene

Fake shadows

- Fake methods

Images from TombRaider. ©Eidos Interactive.

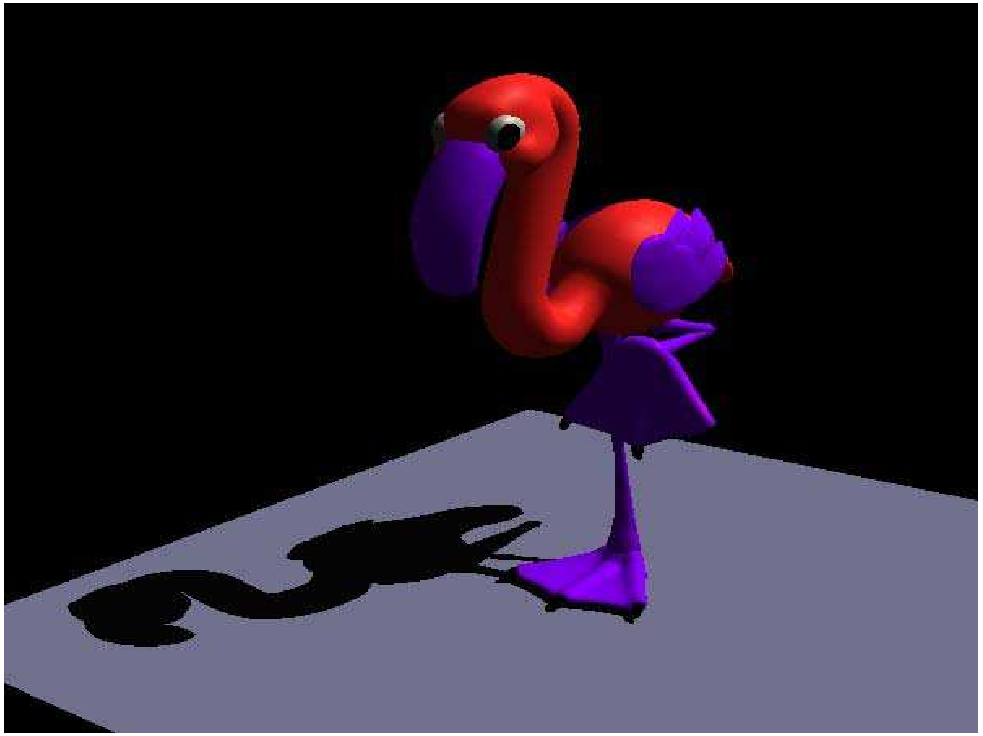
- Simple shadows (elipses, polygons ...)
- Precomputed shape that moves with object

Soft shadows

- Main idea:
  - Using many samples of the light source and export the results to texture

Planar (Projected) Shadows

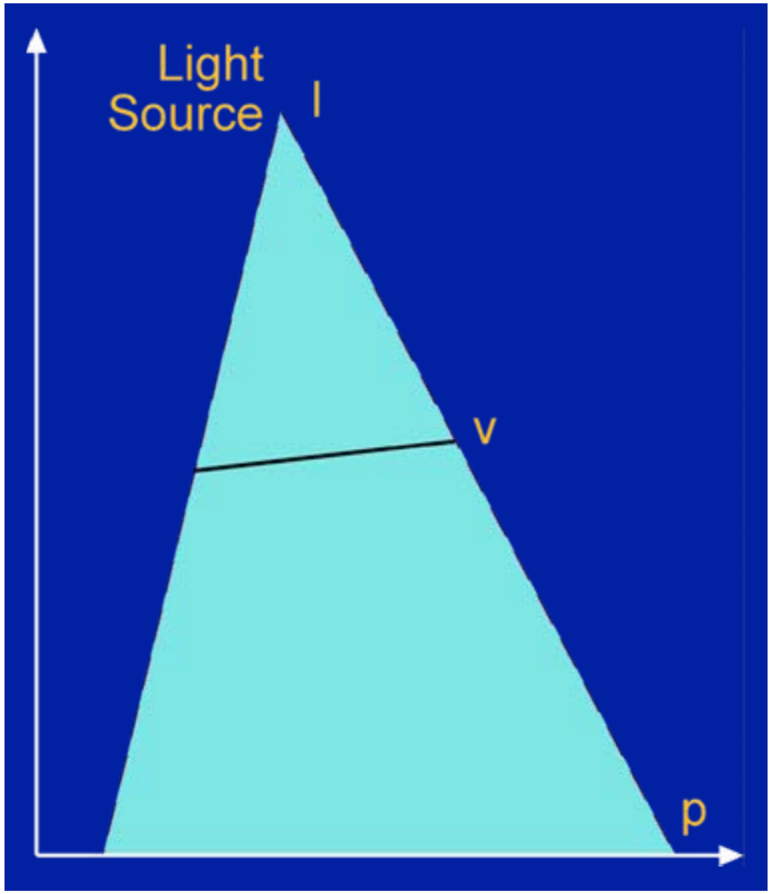
- [Blinn88] *Me and my fake shadow*
  - The shadows are projected onto selected large polygons
    - Ground plane
    - Walls



(c) Stefan Brabec

Projected shadows

- Example: Shadows on the plane *xz*; *y*=0



$$\vec{p} = \vec{l} + t(\vec{v} - \vec{l})$$

$$t = \frac{l_y}{l_y - v_y}$$

Projected shadows

- Transformation is determined by a 4x4 matrix

$$p_x = \frac{l_y v_x - l_x v_y}{l_y - v_y}$$

$$p_z = \frac{l_y v_z - l_z v_y}{l_y - v_y}$$

$$\vec{p} = \begin{pmatrix} l_y & -l_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -l_z & l_y & 0 \\ 0 & -1 & 0 & l_y \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix}$$

Projected shadows

- In general: the receiver is a plane *E*

$$E : \vec{n} \bullet \vec{x} + d = 0$$

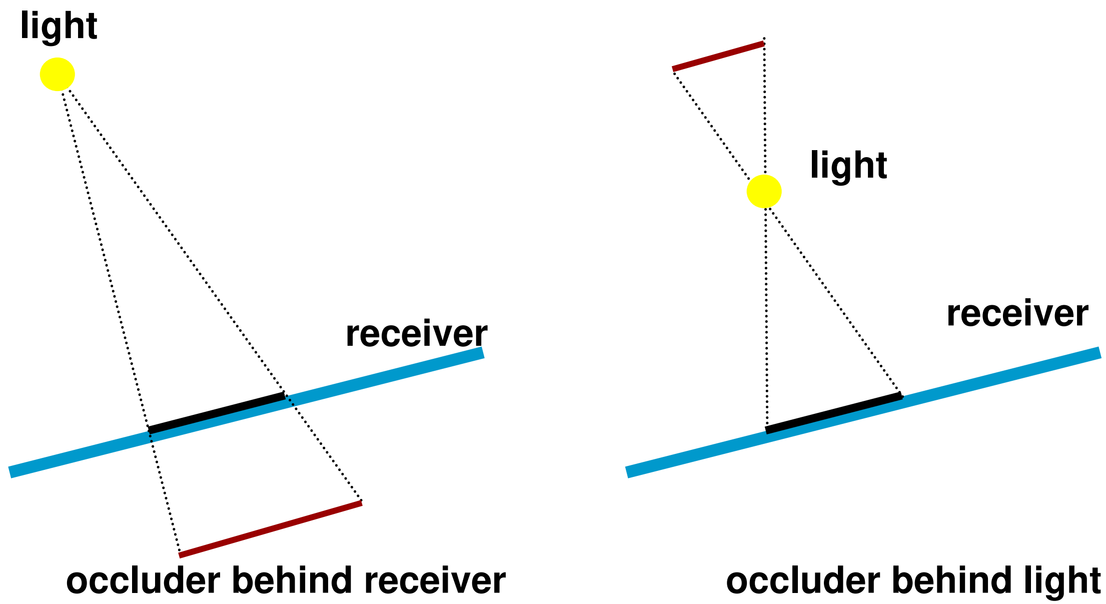
$$\vec{p} = \vec{l} - \frac{d + \vec{n} \bullet \vec{l}}{\vec{n} \bullet \left(\vec{v} - \vec{l}\right)} (\vec{v} - \vec{l})$$

Projected shadows

- Basic algorithm:
  - Render scene (full lighting)
  - For each receiver polygon:
    - Compute projection matrix M
    - Multiply with actual transformation (modelview)
    - Render selected (occluder) geometry
      - Darken a pixel or draw it with black color

Projected shadows - problems

- Wrong Shadows & Anti-Shadows
  Objects behind light source or objects behind receiver
  - If $t < 0$



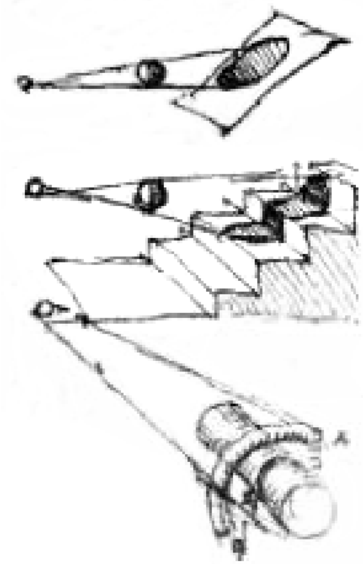
  - Solution is clipping
- Does not produce self-shadows, shadows on curved surfaces, etc.

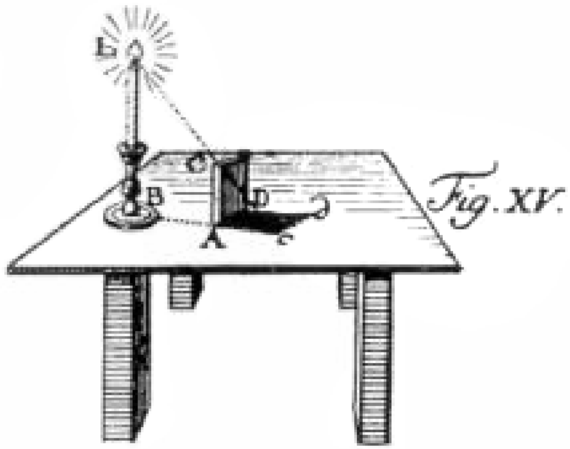
Projected shadows

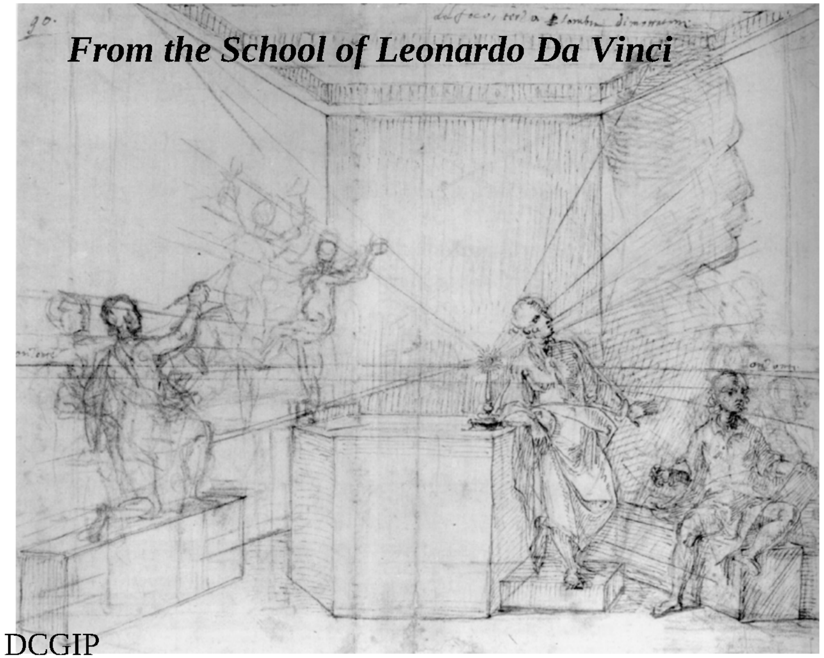
- Summary:
  - Only practical for very few, large receivers
  - Easy to implement
  - Use stencil buffer (z fighting, overlap, receiver)
  - Efficiency can be improved by rendering shadow polygons to texture maps
    - Render selected (occluder) geometry

Shadow volumes

- [Crow77] *Shadow algorithms for computer graphics*
  - Compute regions of shadow in 3D
    - Object-space algorithm
    - Cast shadows onto arbitrary receiver geometry (polygons).



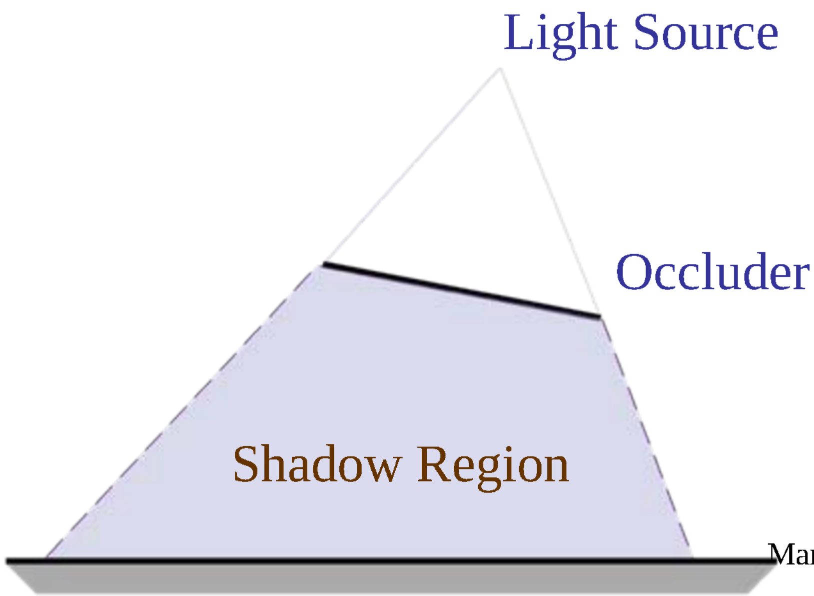


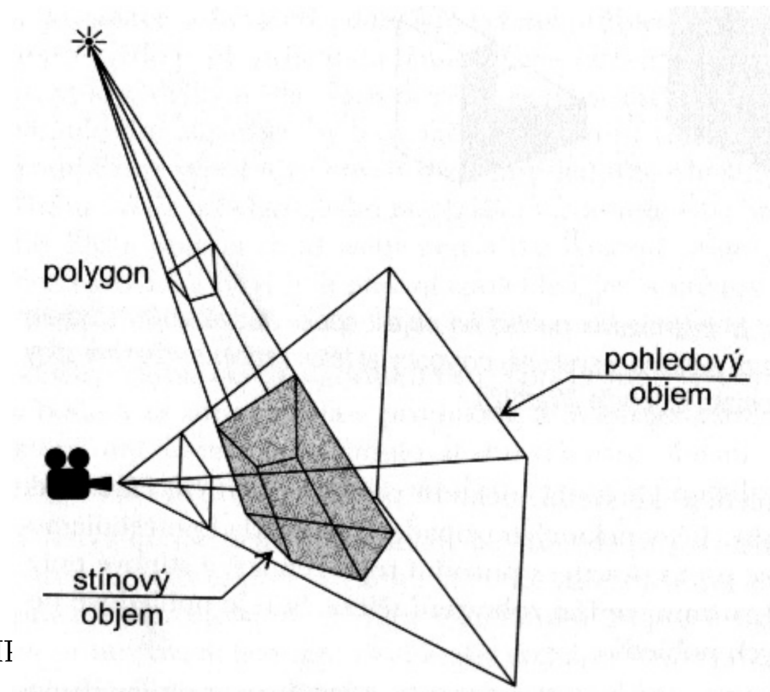


From the School of Leonardo Da Vinci

Shadow volumes

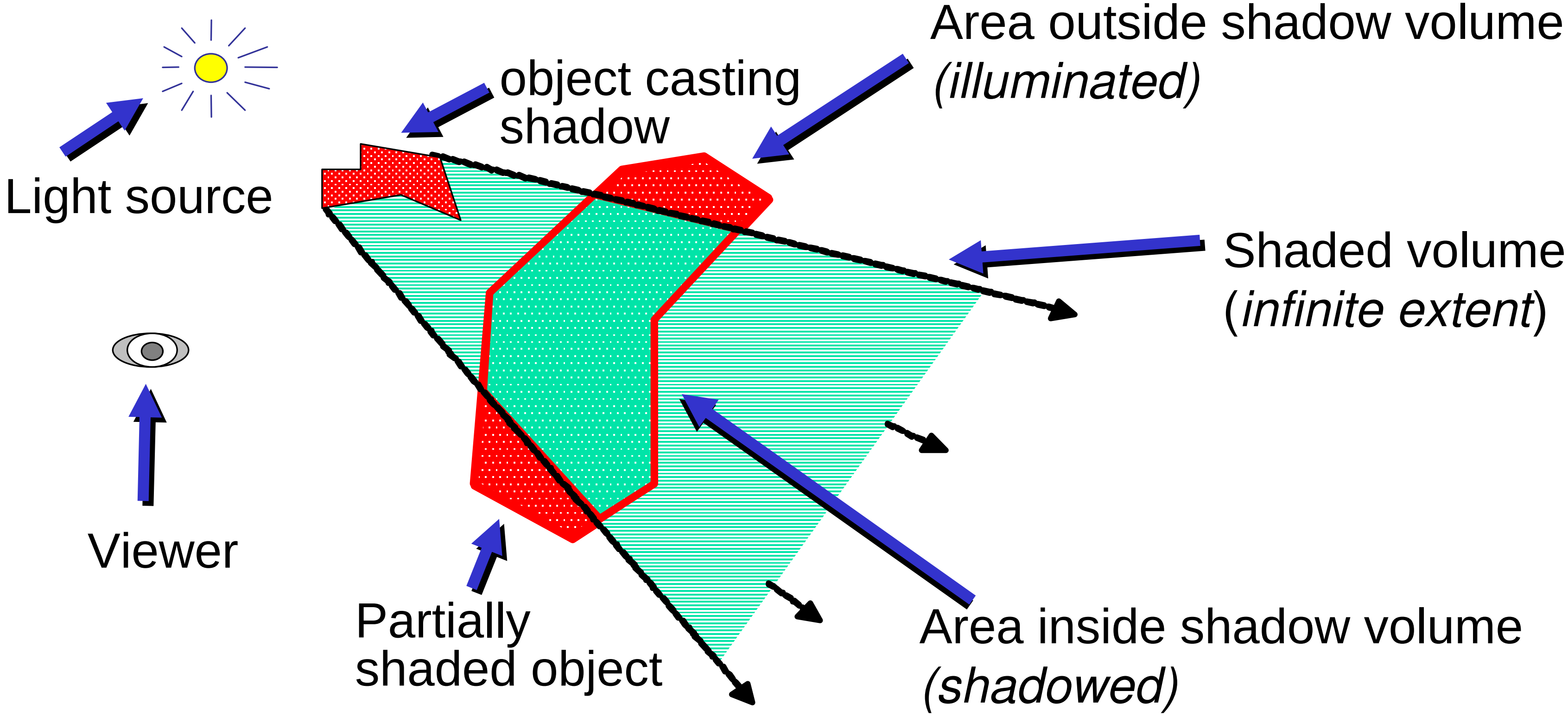
- Basic idea: creating an auxiliary volume
- Extending the shadow polygons into 3D volumes
  - Light source is the *center of the projection*
  - Everything behind the occluder is in the shade
  - Test if the point lies at least in one shadow volume





Shadow volumes

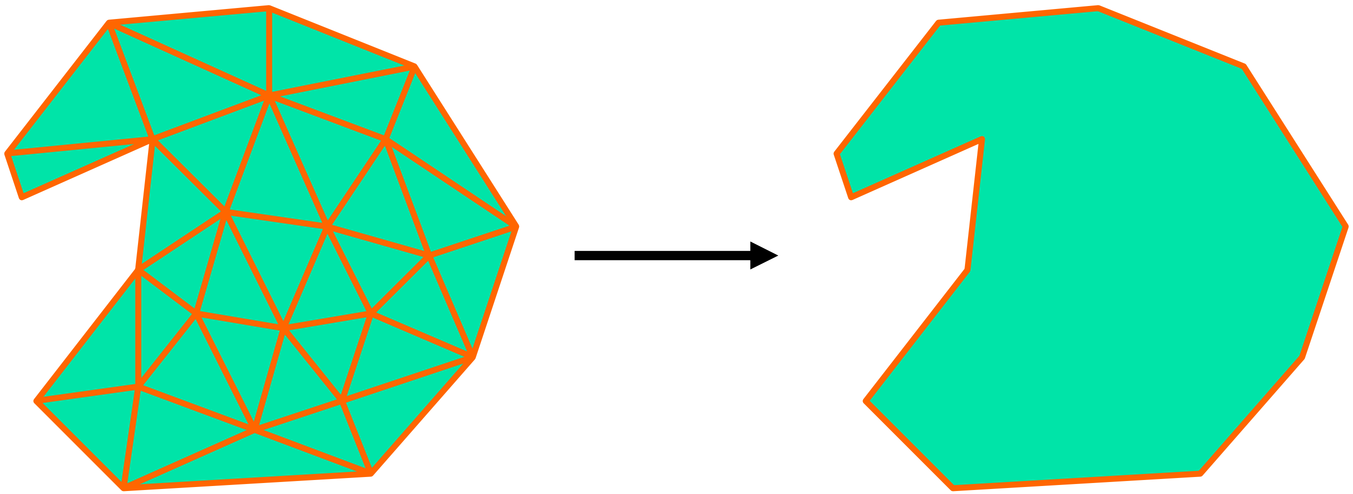
- (2D case)



(c) Stefan Brabec

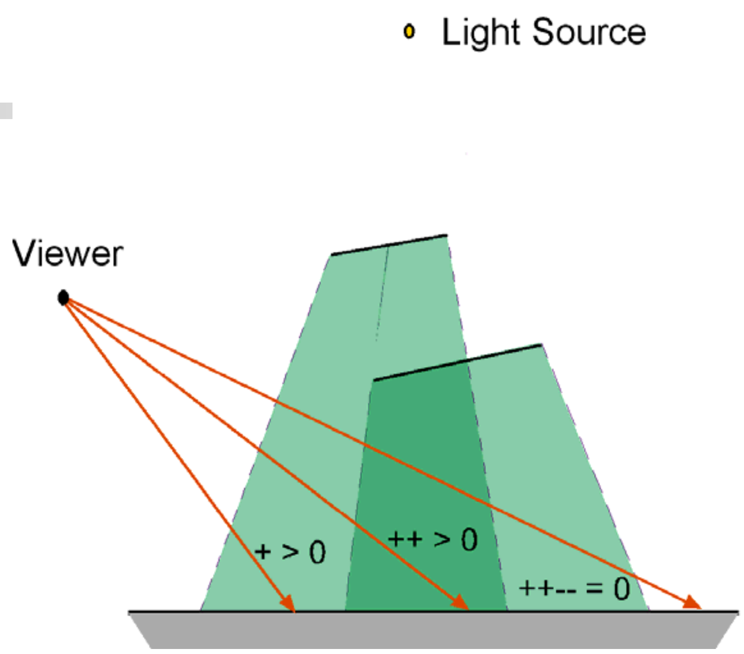
Shadow volumes

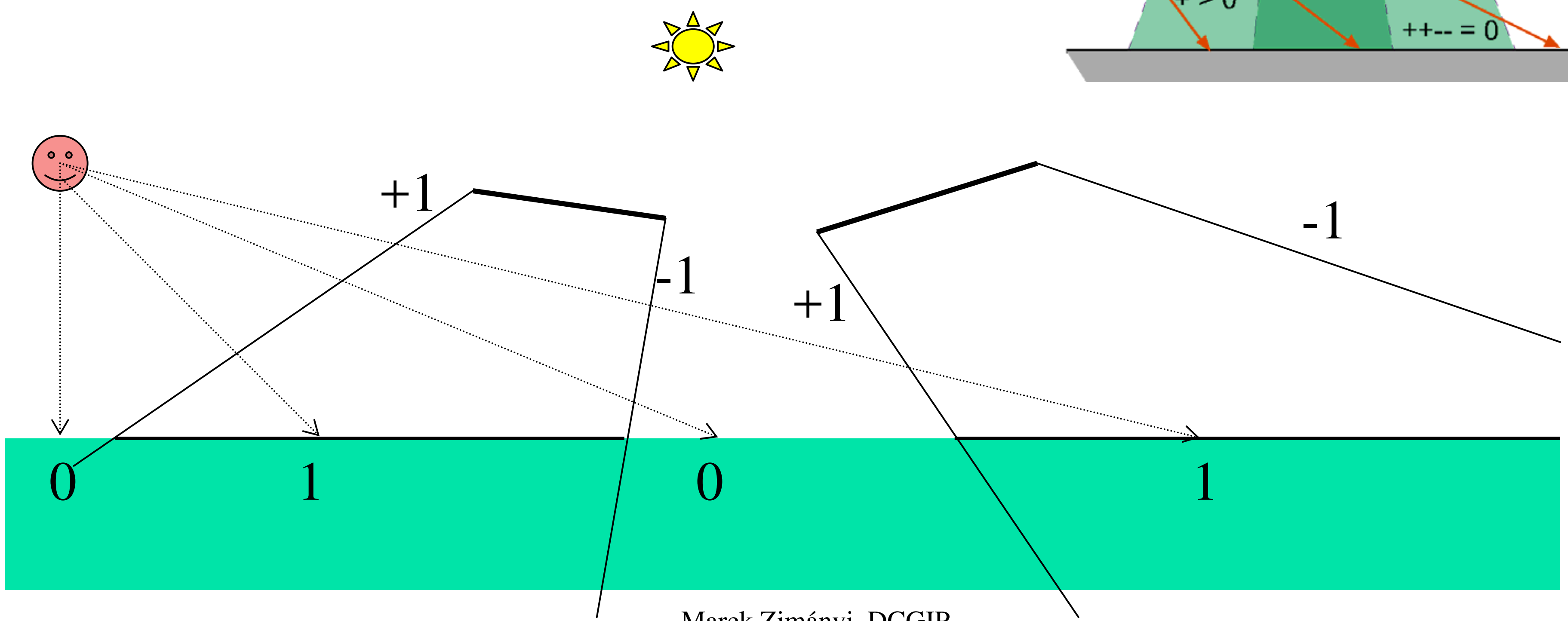
- Creating a shadow volume
  - Simple idea:
    - One shadow volume for every polygon
  - Improvement:
    - Using silhouettes
      - Advantage: Only one shadow volume



Shadow volumes

- Shadow test: *in-out* counter
  - Counter = 0: illuminated
  - Counter > 0: shaded





Shadow volumes

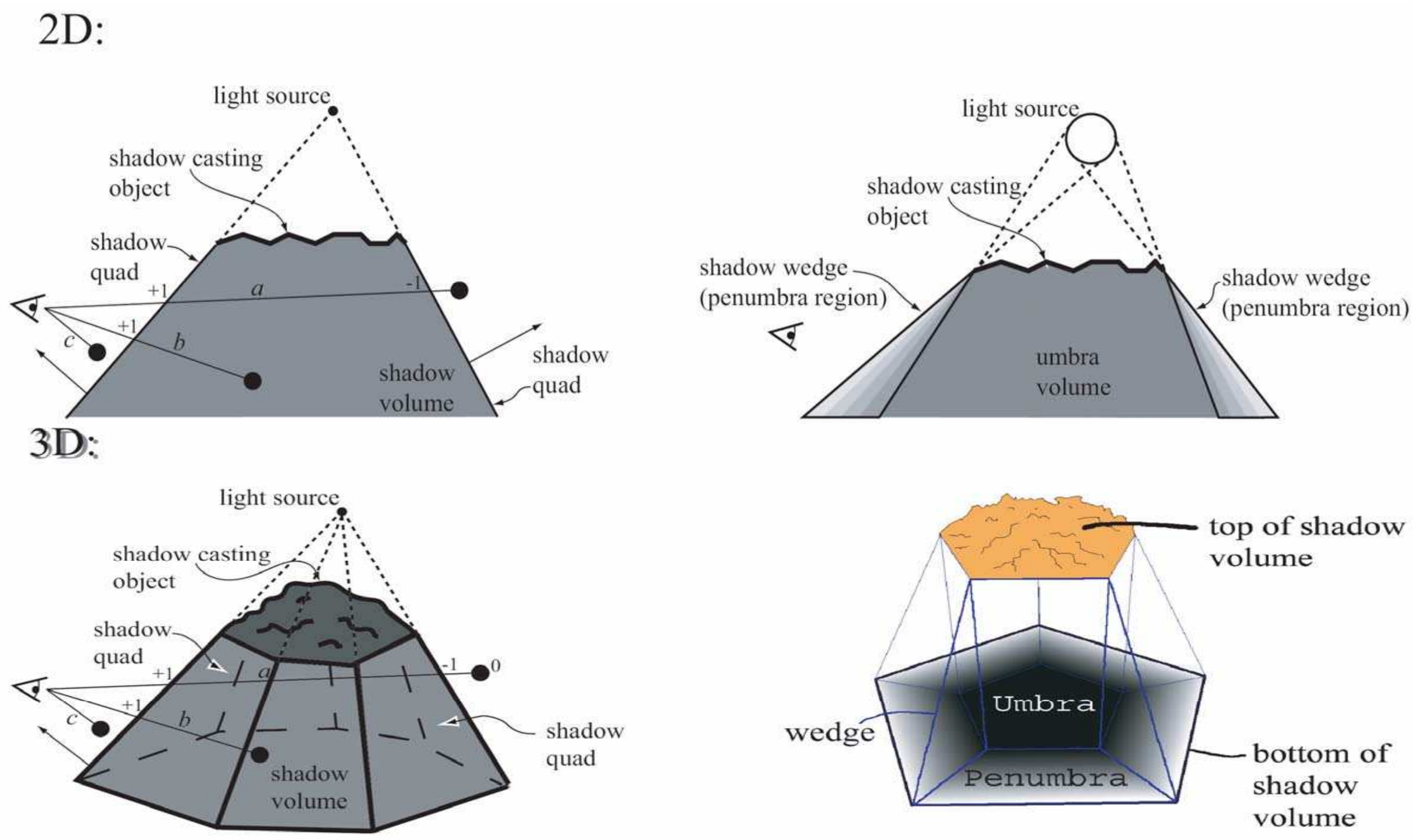
- Algorithm overview (with stencil buffer):
  - Calculation of the shadow volumes
    - View independent!
  - Initialize stencil buffer to 0
  - Turn off frame buffer & z-buffer updates
  - Draw scene with ambient light only
  - Draw front-facing shadow polygons
    - Visible polygons increment the pixel stencil buffer counter
  - Draw back-facing shadow polygons
    - Visible polygons decrement the pixel stencil buffer counter
  - Turn on frame buffer updates
  - Turn on lighting and redraw pixels with counter = 0

Shadow volumes

- Pros
  - Precise, robust
  - For all light sources
  - Extendable to produce soft shadows
- Cons
  - Introduces a lot of new geometry
  - Computational complexity

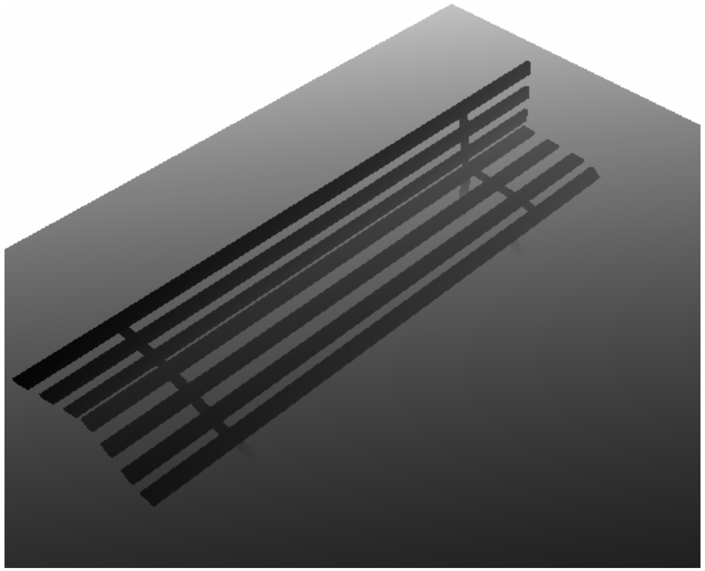
Shadow volumes with soft shadows

-Shadow volumes are created for all vertices of area light sources

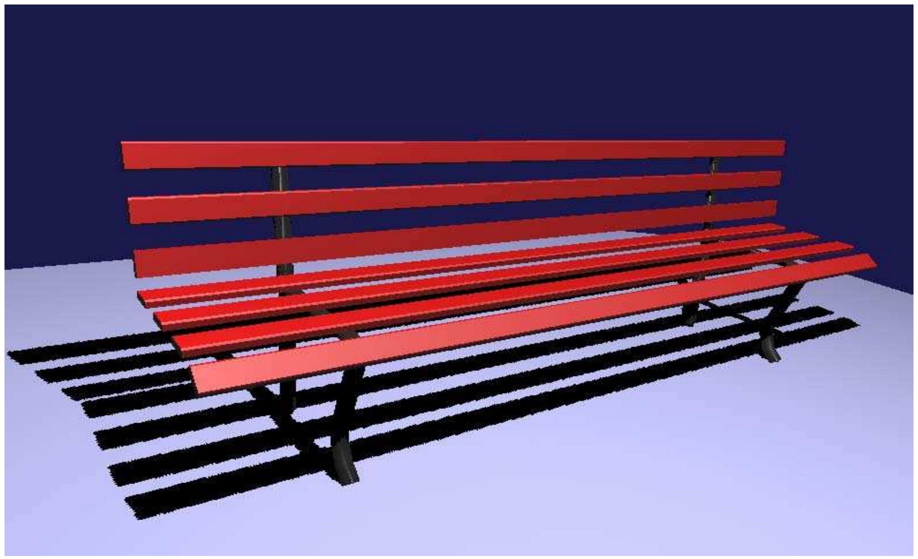


Shadow Maps

- Well suited for hardware implementation
- Uses texture mapping with depth information
- Requires >= 2 passes through the pipeline
  - Compute shadow map (depth from light source)
  - Render final image (check shadow map to see if points are in shadow)



Shadow map



Final render

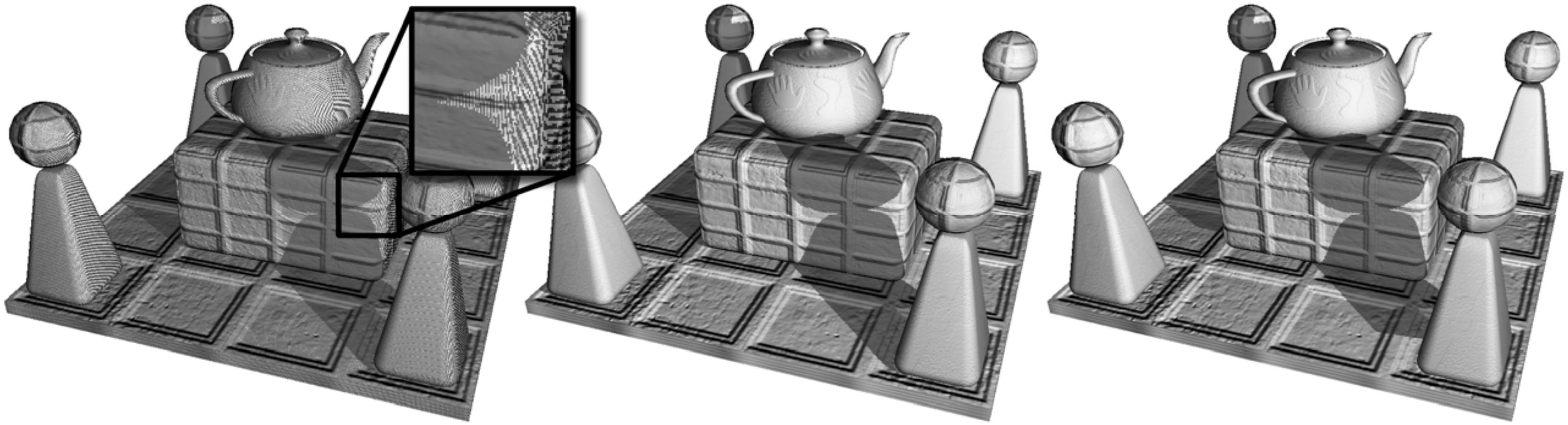
(c) Stefan Brabec

Shadow mapping

1. Render scene as seen from light source L_i
   Save back depth buffer (2D shadow map) to H_i
2. Render scene from viewer's position using depth buffer
3. For all pixels [u,v] (with depth w) of the rendered scene:
   (a) Transform pixel coordinates [u,v,w] to light source space L_i => [x,y,z]
   (b) $A = H_i[x,y]$; $B = z$
   (c) Compare A value stored in shadow map with B:
   If ($A<B$) than the pixel [u,v] is in the shadow, otherwise pixel is illuminated by L_i

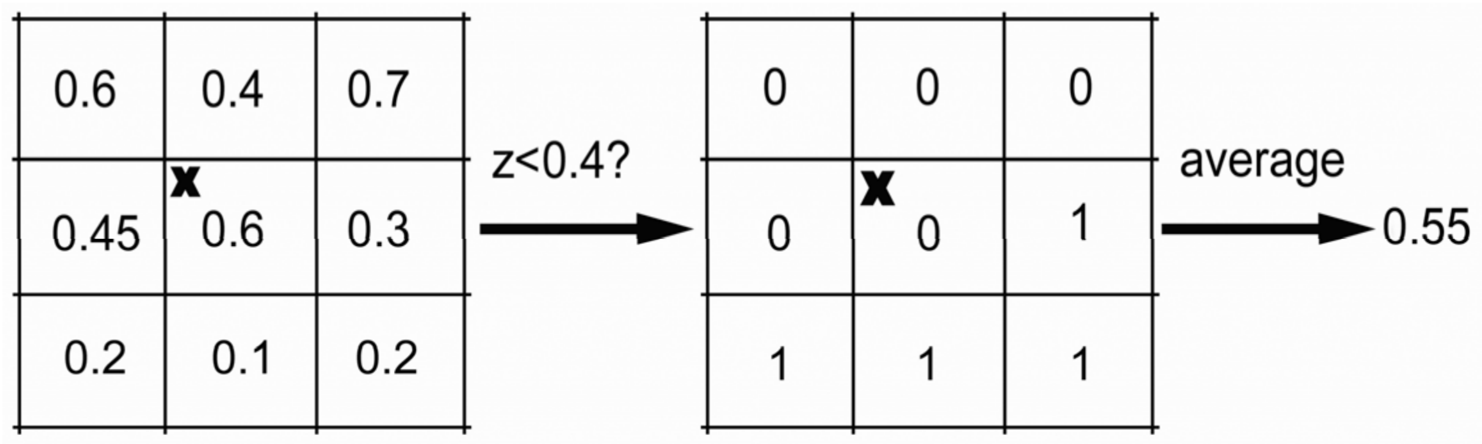
Shadow Maps

- Errors when comparing the depth:
  - "Depth bias" = problem of rounding 'z' coordinate after transformation
  - It can be limited by adding a small value (depth bias) to the stored distances or storing the average of the distance of the two closest points
  - Better solution: In the z-buffer draw only the faces away from the light
    - Allows the use of textures with a lower bit/pixel value
    - It can be combined with the previous solution



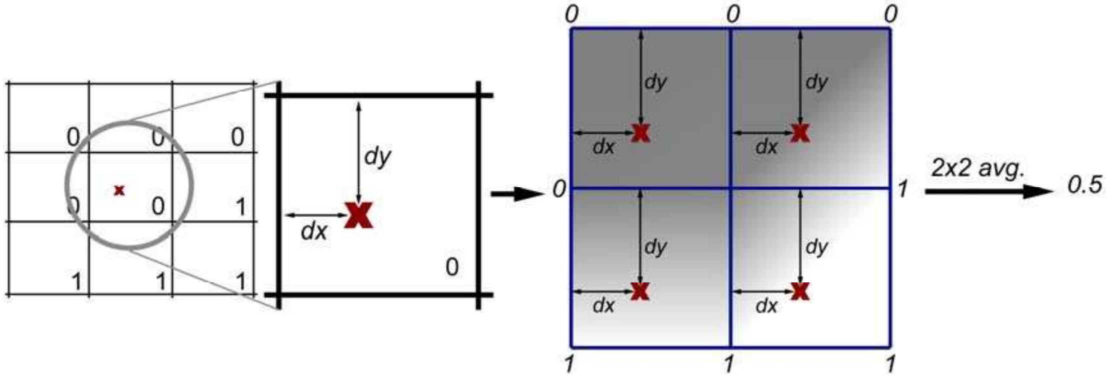
Shadow Maps

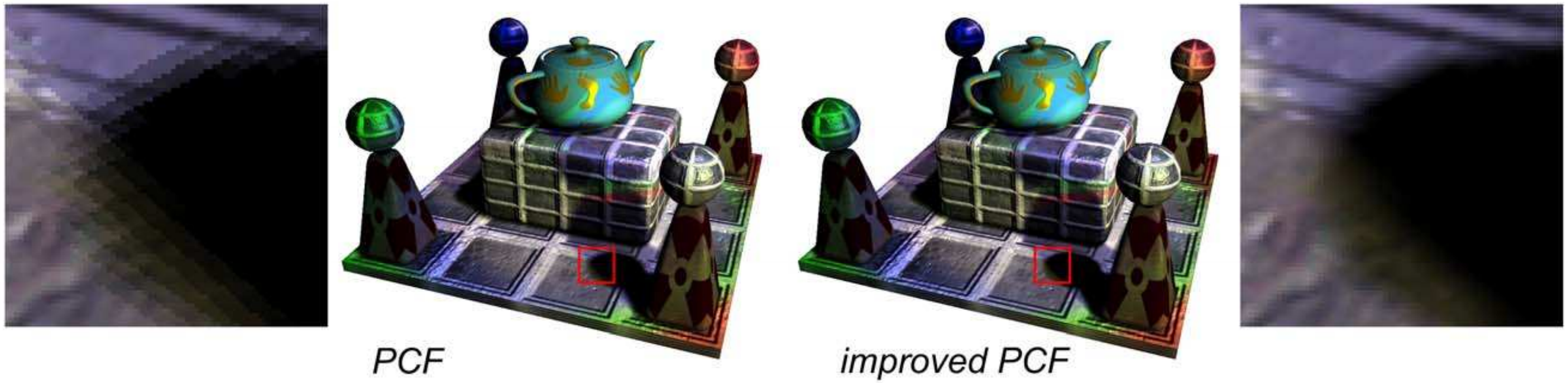
- Errors caused by z-buffers:
  - Aliased shadows
  - Filtering depth values makes no sense
  - Percentage closer filtering - PCF (Reeves1987)
    - first the depth is compared in a certain area
    - the average value is calculated from the generated binary map



Shadow Maps

- Billinear filtered PCF
  - Inserts intermediate step with bilinear filtration

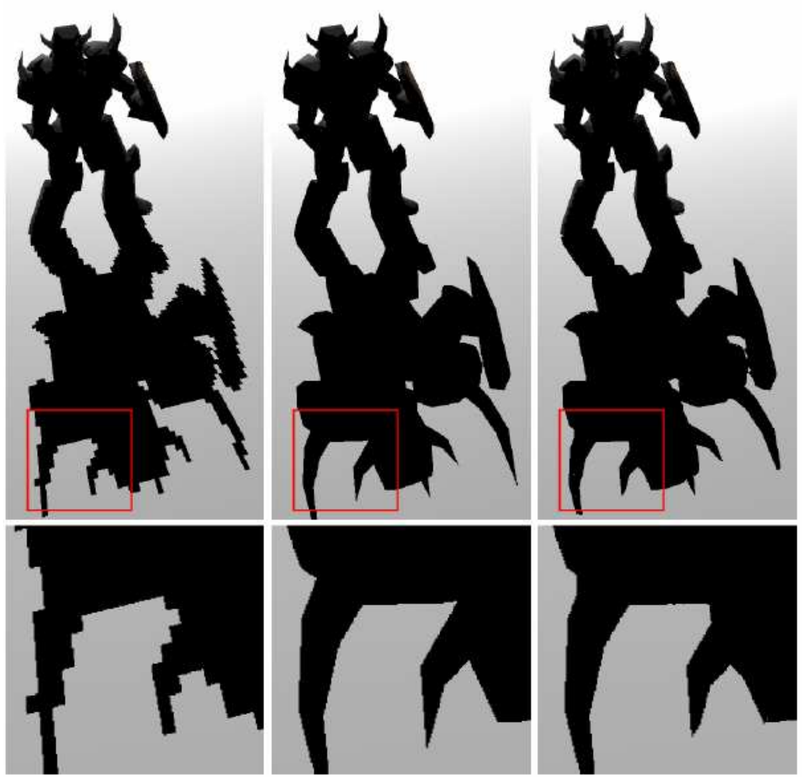




Shadow maps - example

Shadow Silouette Maps

- Pradeep Sen, Mike Cammarano and Pat Hanrahan
- Extension of the shadow mapping
  - Approximate object silhouette



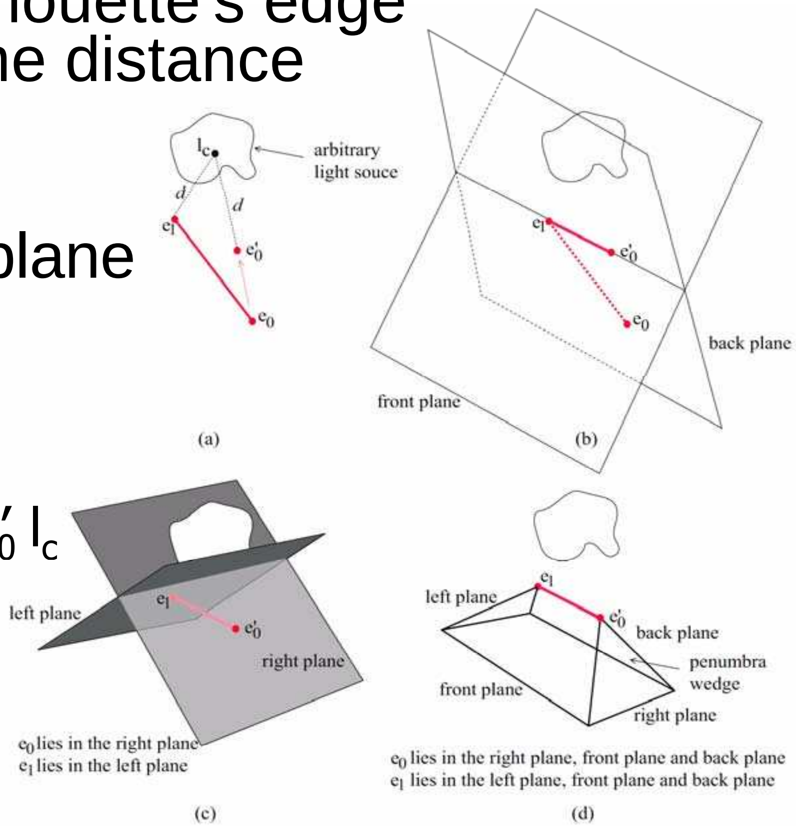
Construction of wedges

a) The outermost edge of the silhouette's edge from the light shifts to the same distance as the closer

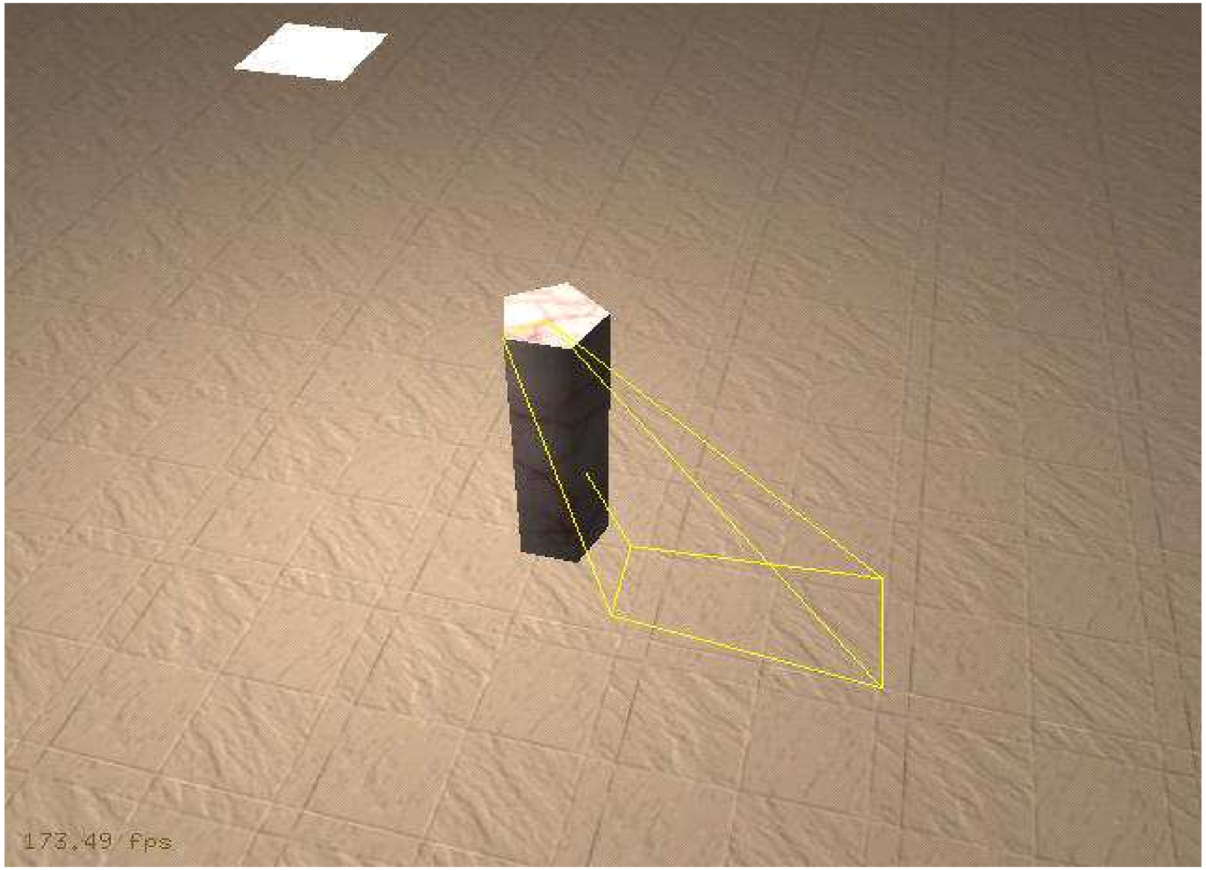
b) Creation of the front and rear plane

c) Right plane
- contains e'_0
- perpendicular vector to $e_1e'_0$ and $e'_0 l_c$
- "touches" the light surface
- left plane similarly

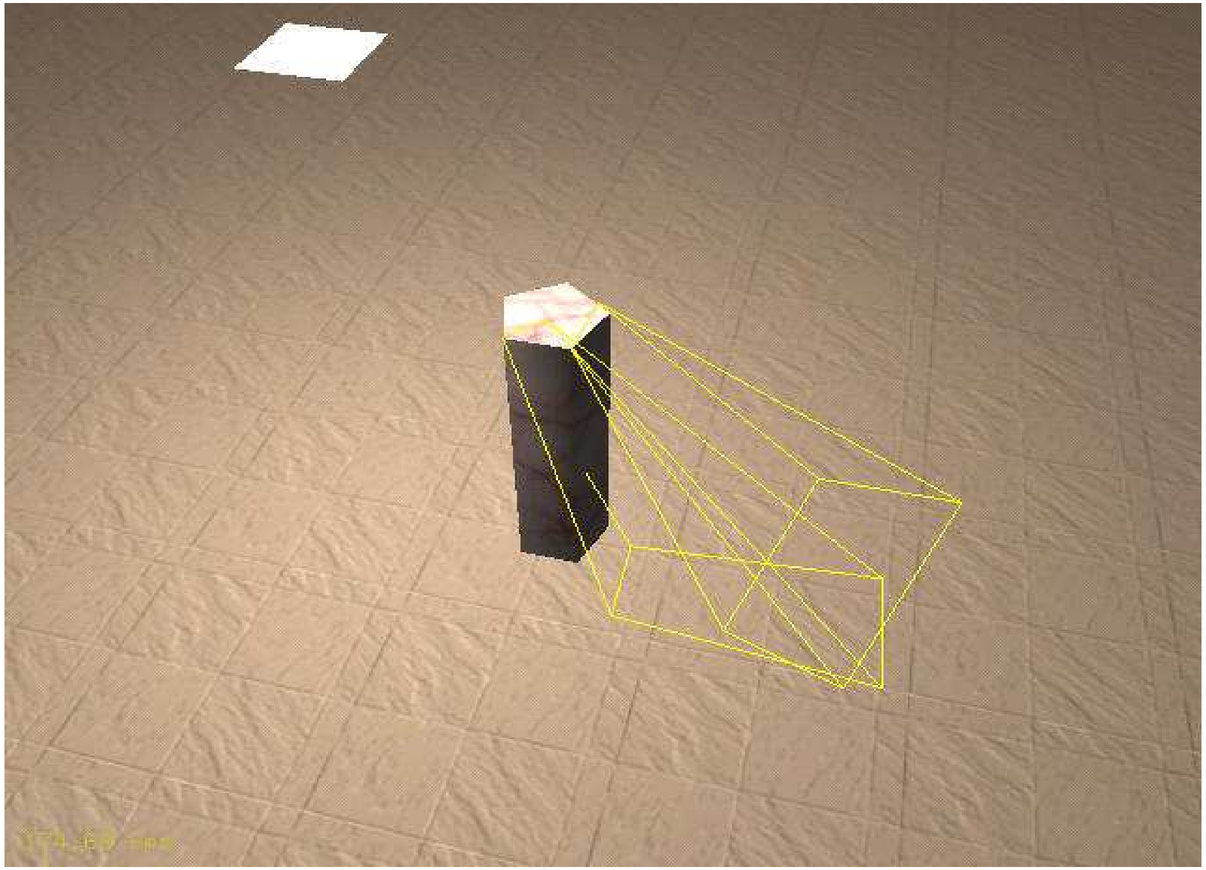
d) Final wedge



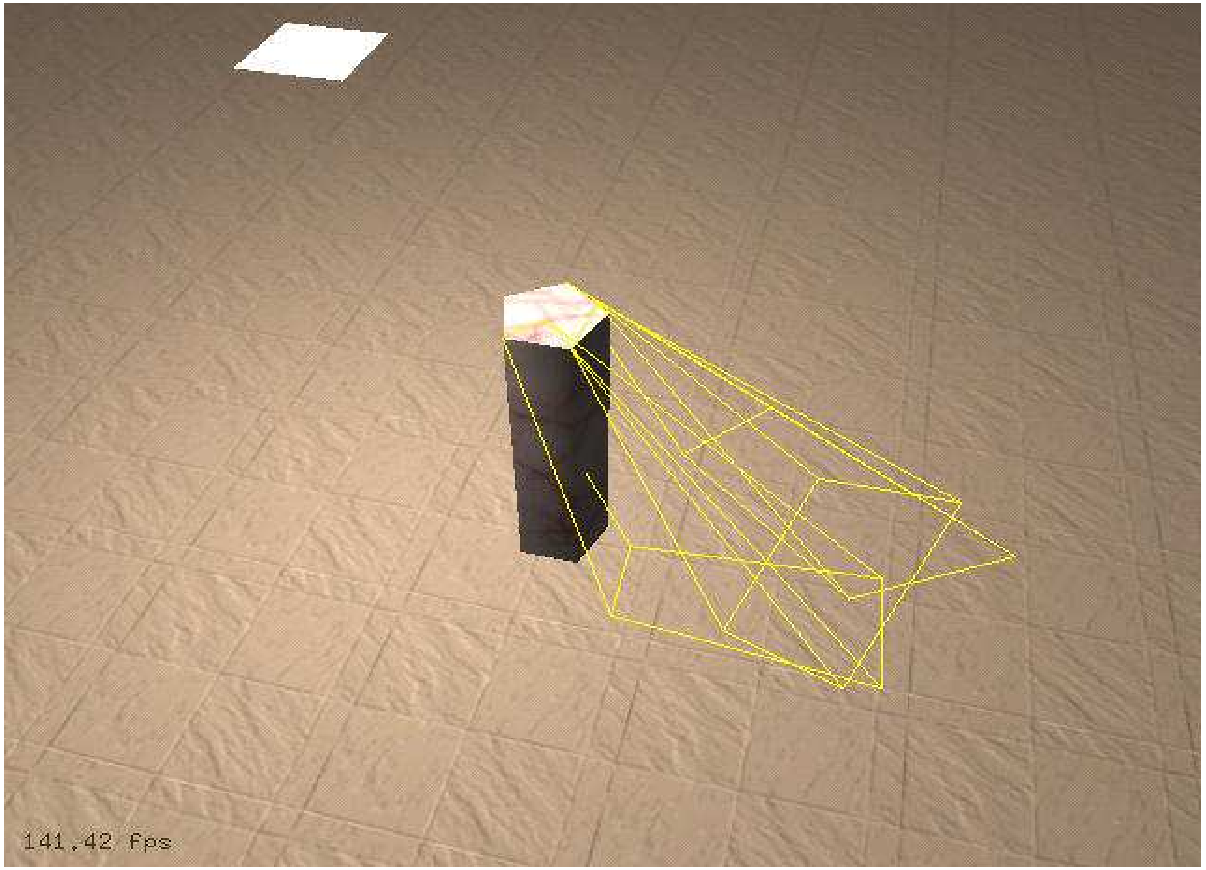
Construction of wedges



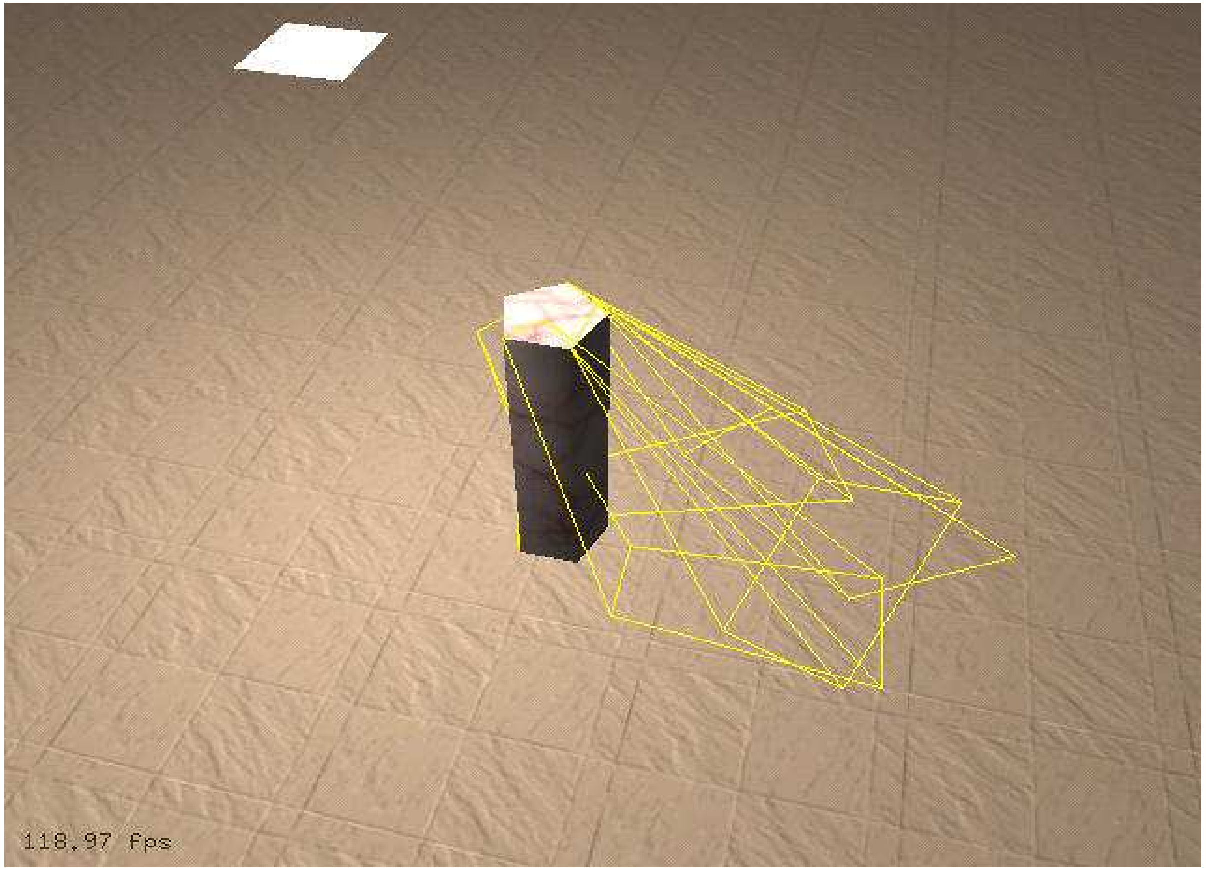
Construction of wedges



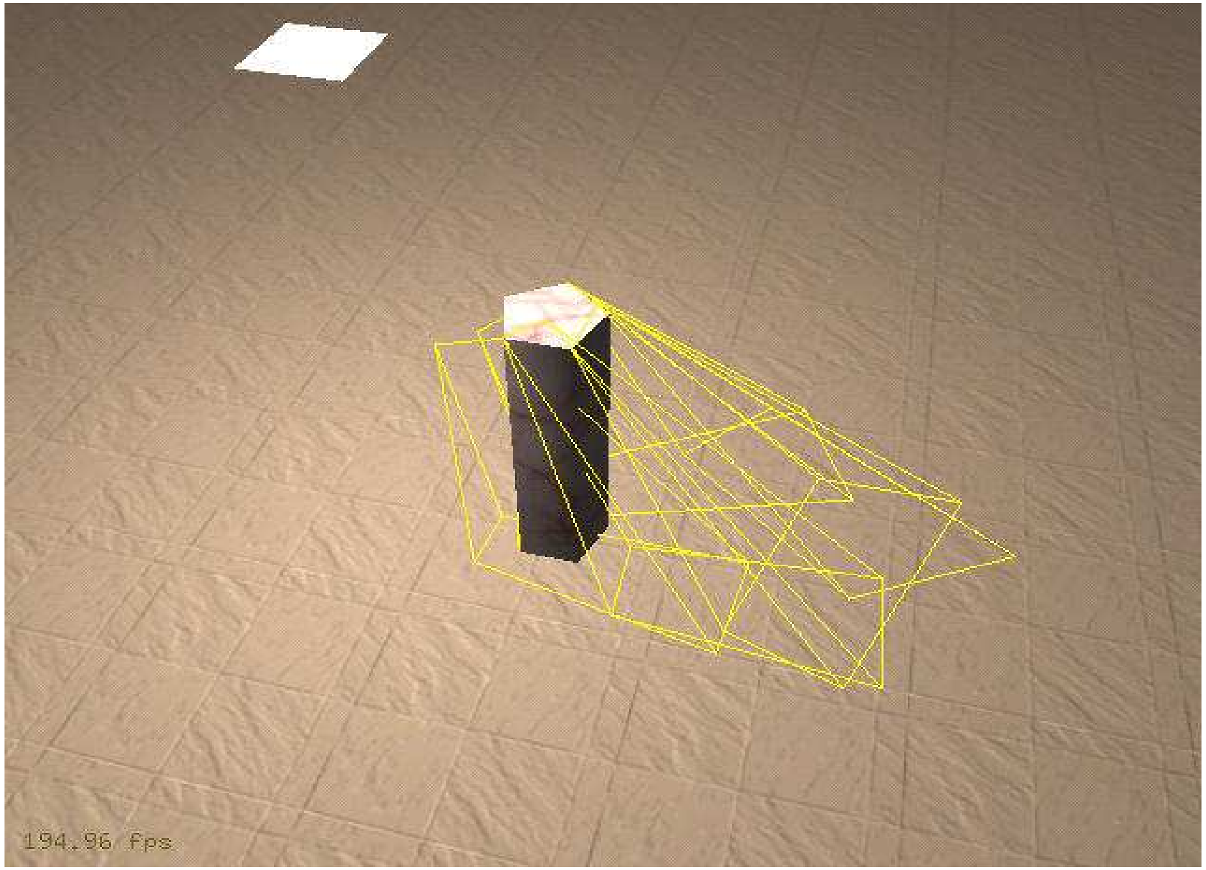
Construction of wedges



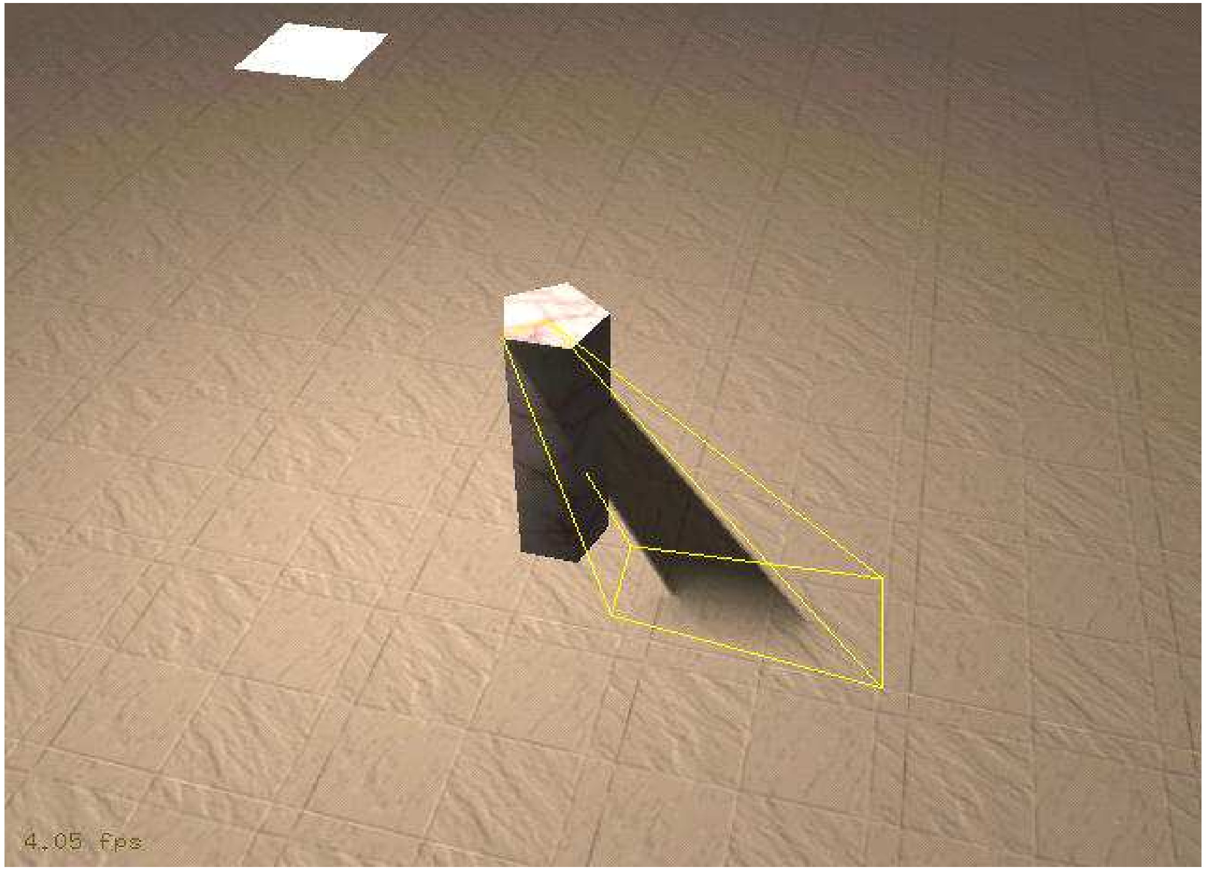
Construction of wedges



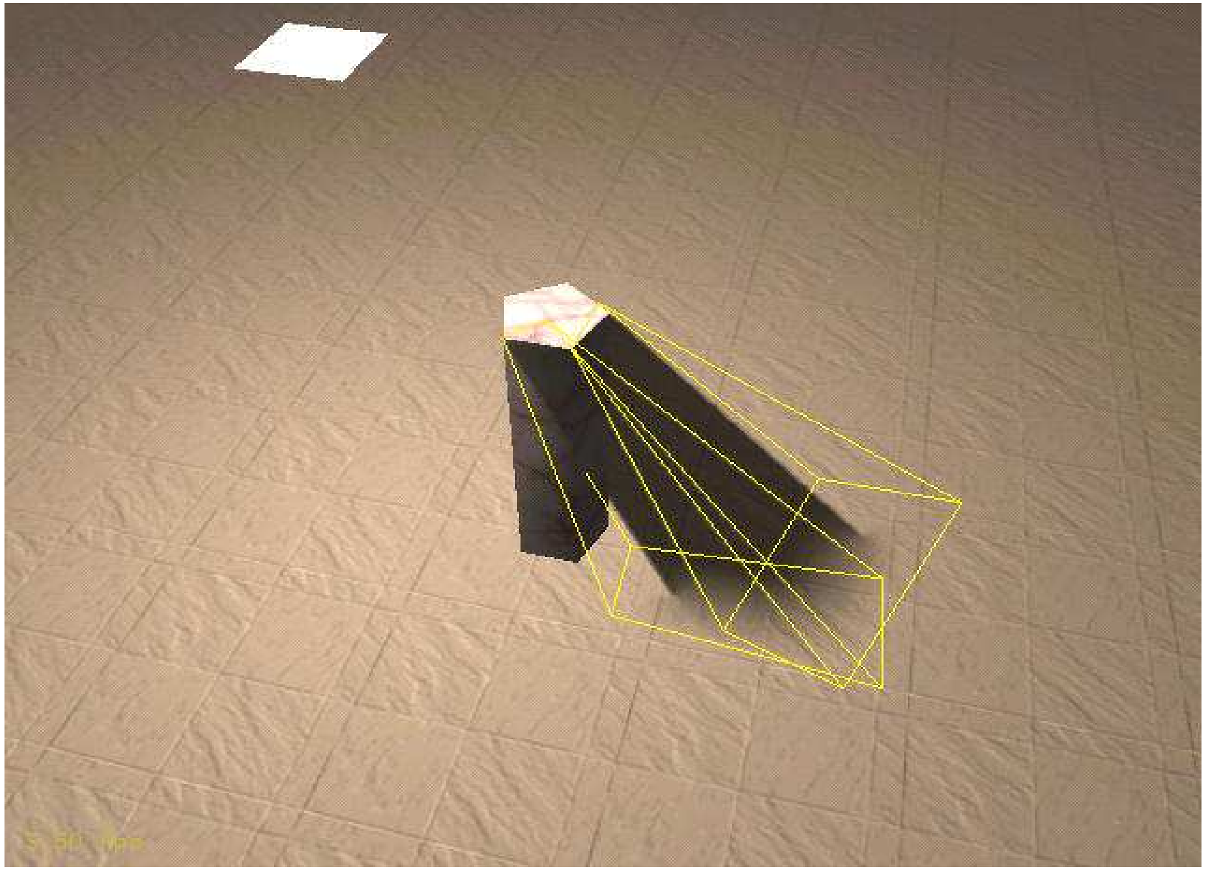
Construction of wedges



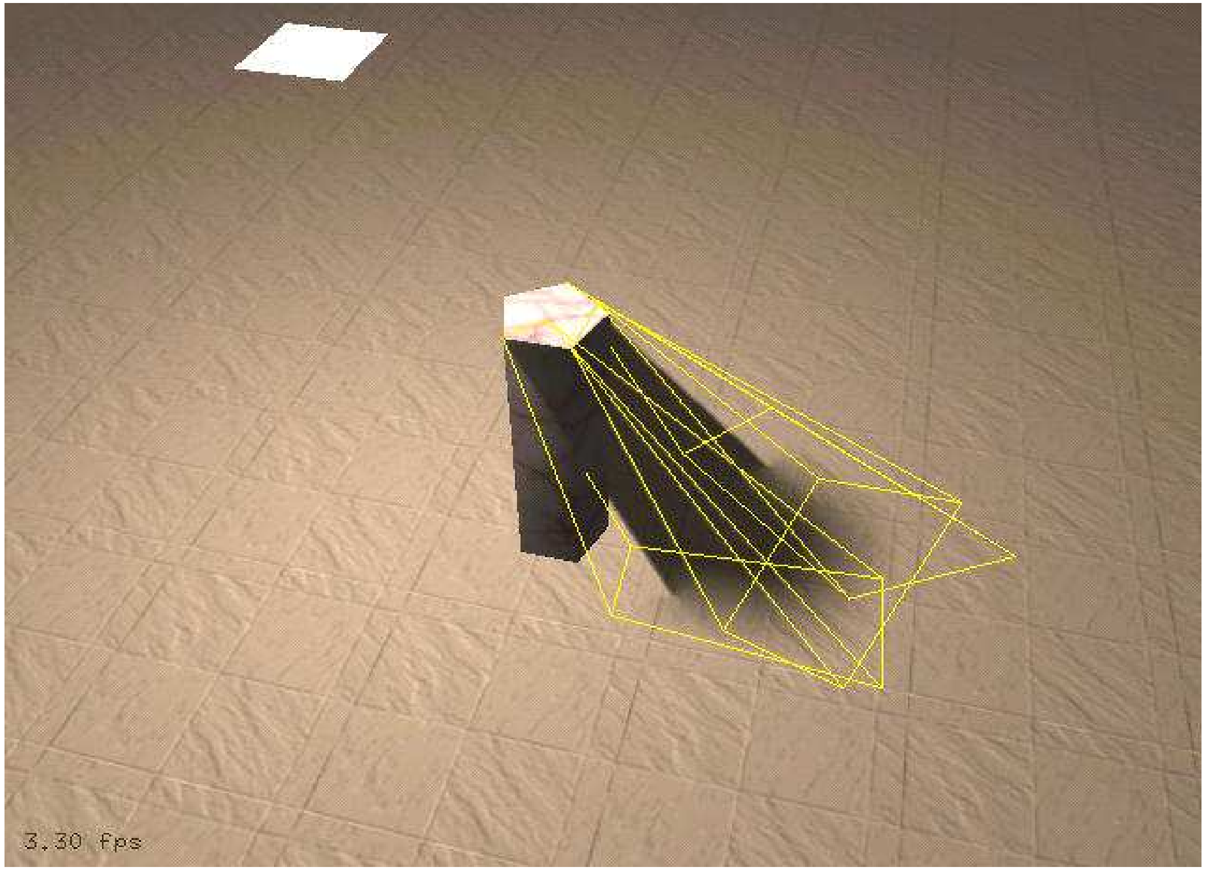
Rasterization



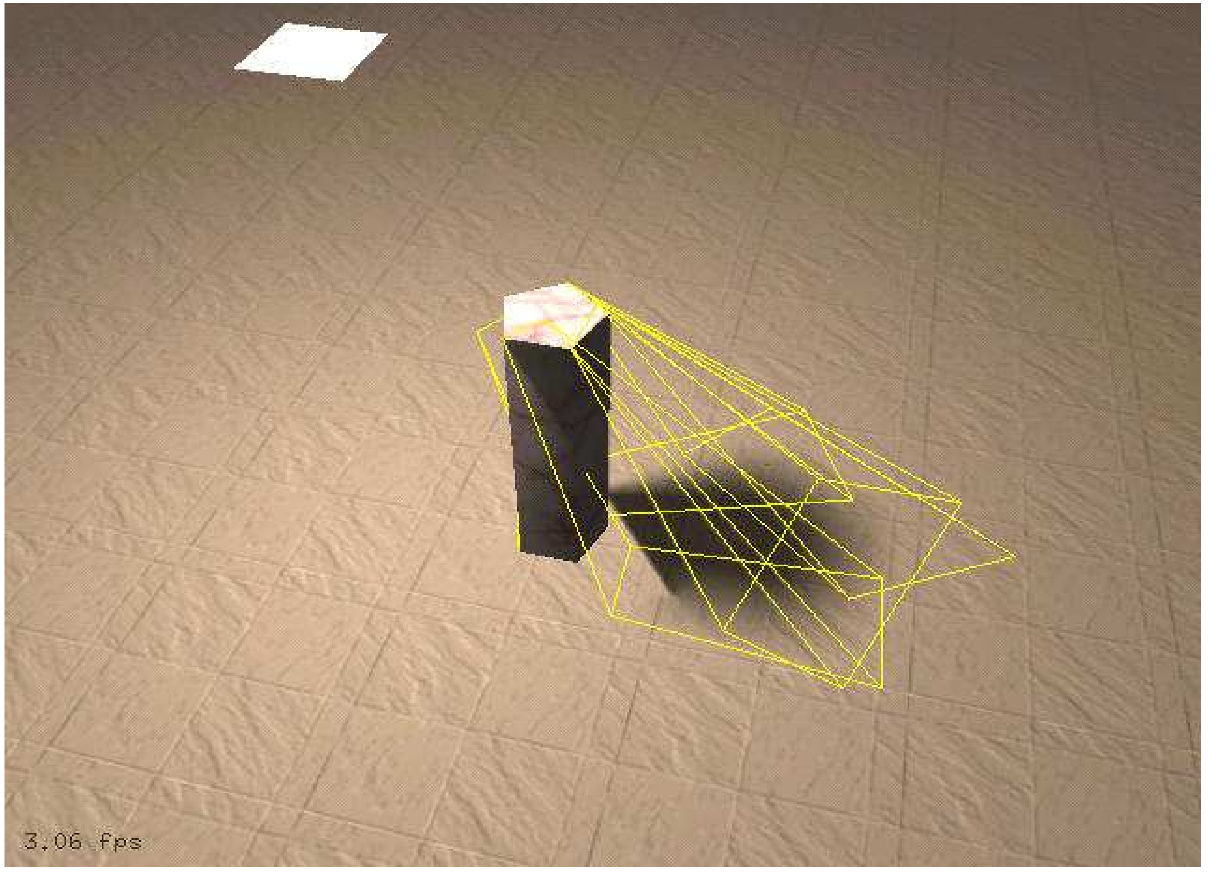
Rasterization



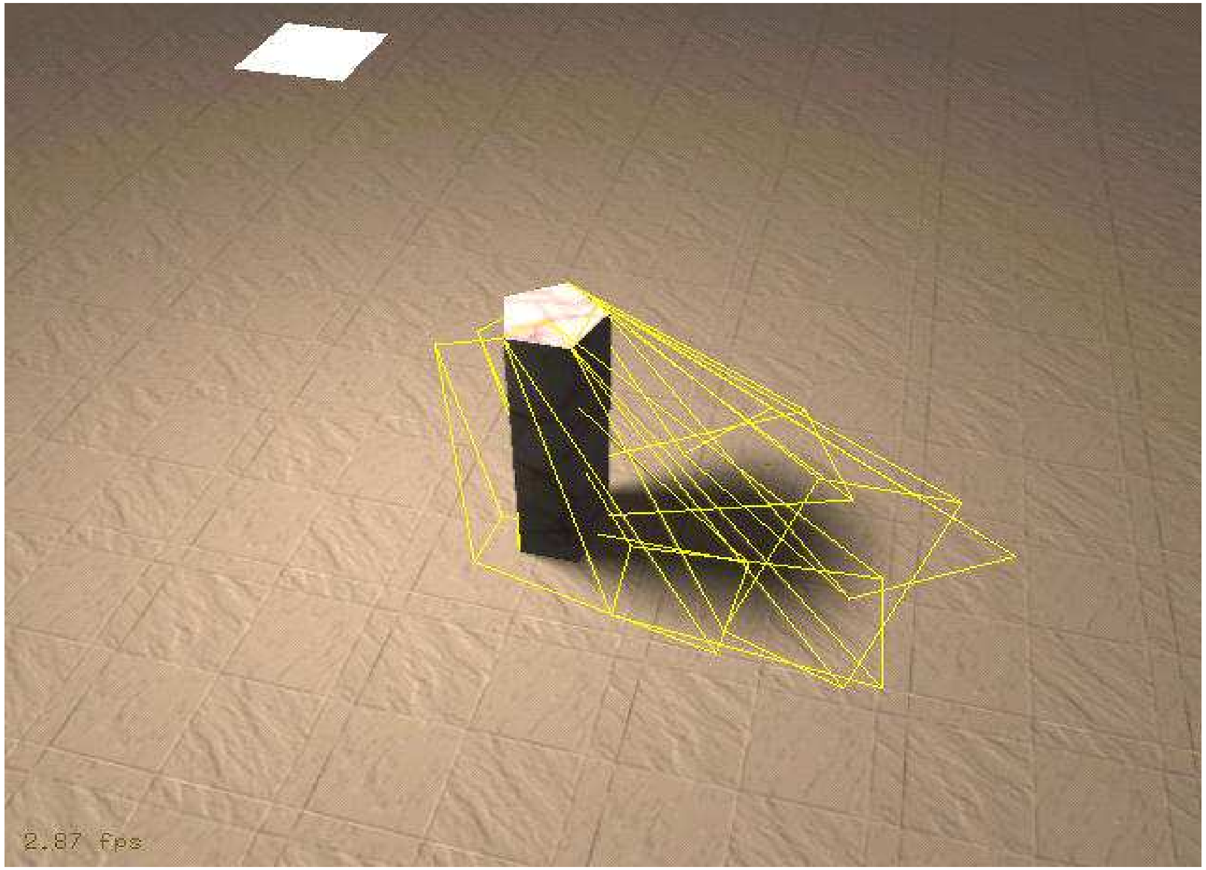
Rasterization



Rasterization



Rasterization



Construction of wedges

- Pros
  - Independence of individual wedges
  - Speed
- Cons
  - Large at the edges with greater distances
    - Computational complexity