Beyond simple shading…

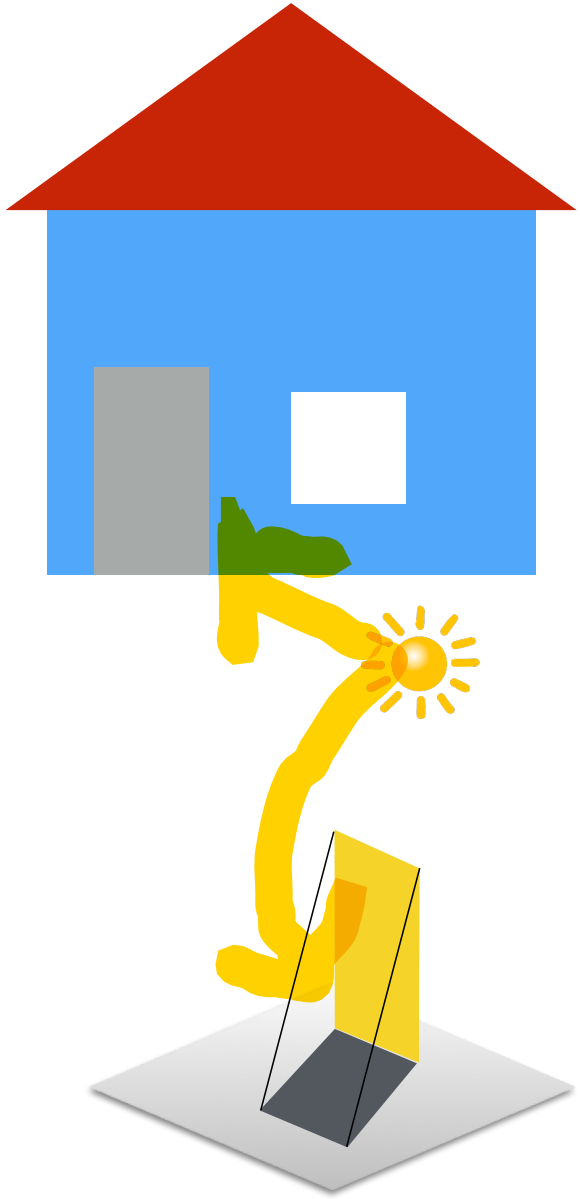
Surface Detail

- Most real surfaces don't look smooth or uniform
- Surface detail is missing!
- Several techniques can be used to add detail to surfaces…

Surface Detail

Surface-Detail Polygons

- create features on surfaces by adding polygons (doors, windows, lettering, shadows) to a base polygon
- surface-detail polygons need to be given priority in visible surface determination
- surface-detail polygons are rendered and made visible instead of the base polygon, thus adding detail



Surface Detail

Texture Mapping

How many surface-detail polygons would be needed for the image below?

The finer the detail, the less viable it is to add surface-detail polygons…

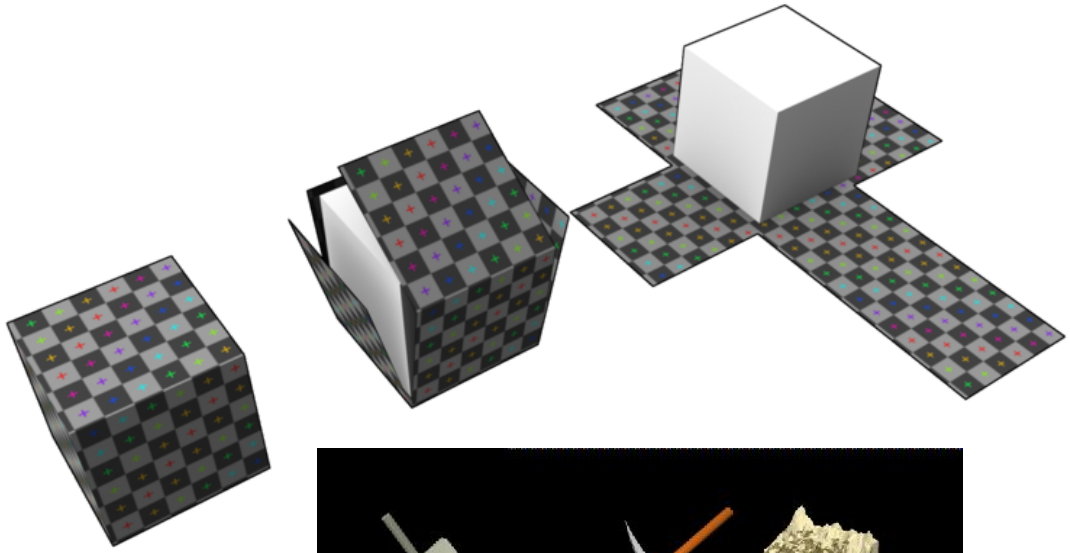
Better map an image (digitised or synthesised) onto the surfaces

*from Unity3D Forum

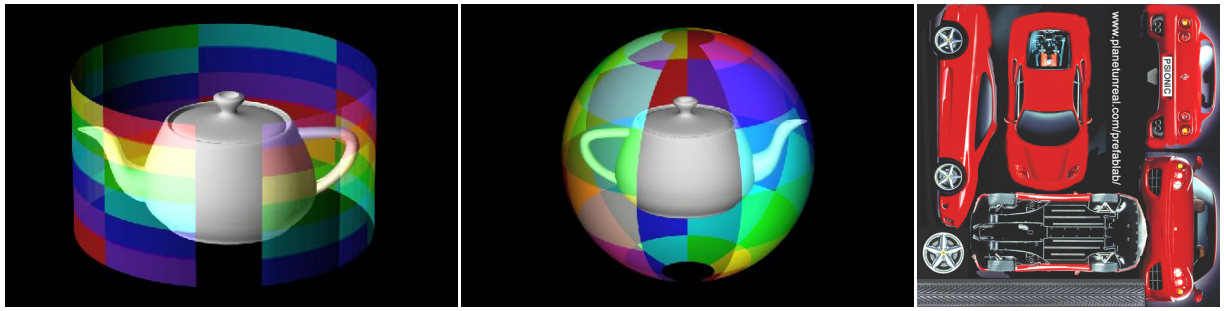
Surface Detail

Texture Mapping

- Mapping can be done in 2D (painting) or 3D (sculpting).
- Textures can be digitised or synthesised
- Textures can be used for almost anything related to shading, and not only to replace object's colour!
- What type of mapping to use?
- What will we do with the looked up value?



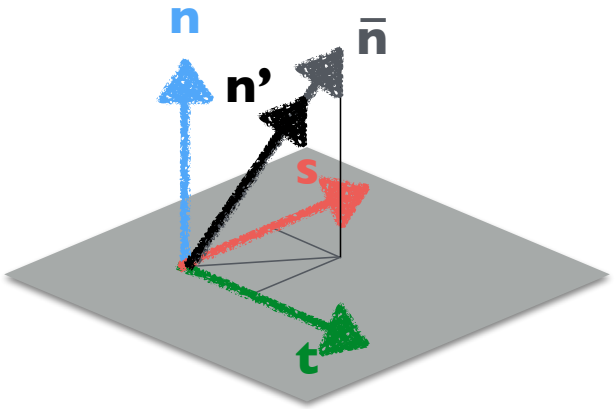
*from SIGGRAPH.org education materials



Surface Detail

Bump mapping

- Bump mapping perturbs surface normals according to a bump map (texture)
- Surface geometry is not perturbed
- Bump maps are implementation dependent. For instance, we can use just the R and G values to tilt the normal vector
- We will need a pair of mutually perpendicular vectors **s** and **t**



$$\overline{\mathbf{n}} = \mathbf{n} + r\mathbf{s} + g\mathbf{t}$$

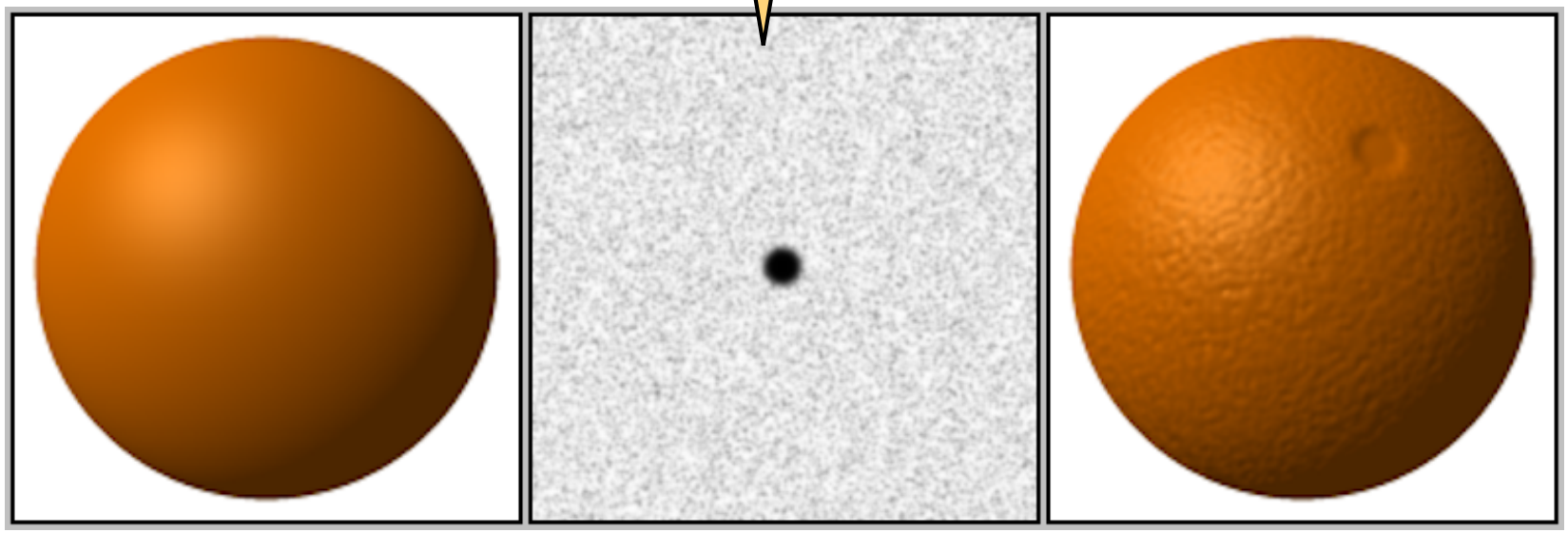
Perturbed normalised normal: $\mathbf{n}' = \frac{\overline{\mathbf{n}}}{\|\overline{\mathbf{n}}\|}$

r and g are the values obtained from the Red and Green channels of the bump texture and represent the directional derivatives of the normal vector along **s** and **t**. Bump map values need to be mapped to [-1,1] range.

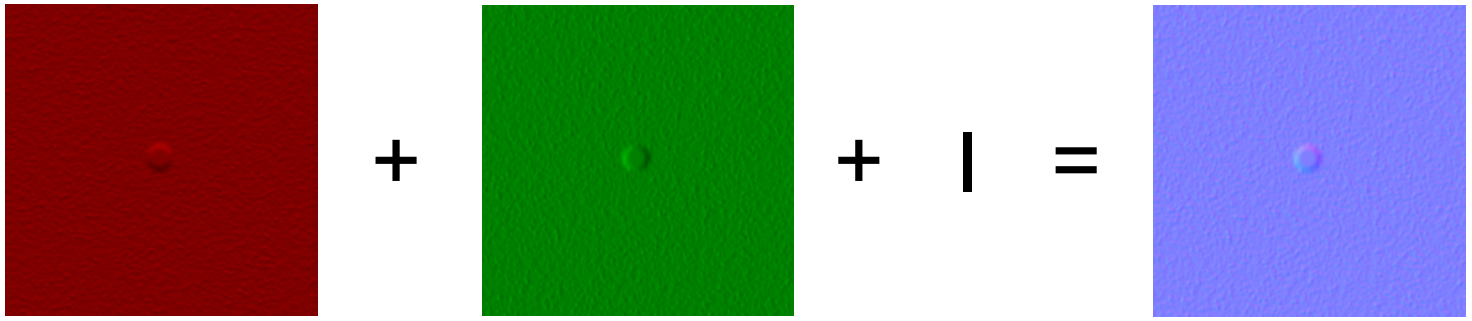
Surface Detail

Bump mapping

Displacement map D - contains displacements along the original normal. Black (0) is mapped to a maximum negative displacement along the normal and white (1) means the opposite, positive displacement. 0.5 grey values mean no displacement.



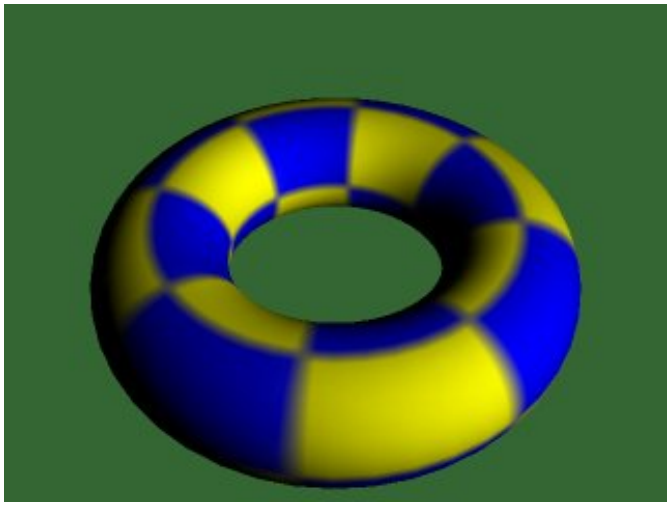
Red = dD/ds + Green = dD/dt + Blue = 1 = Final Bump Map*



* this is just a possible example

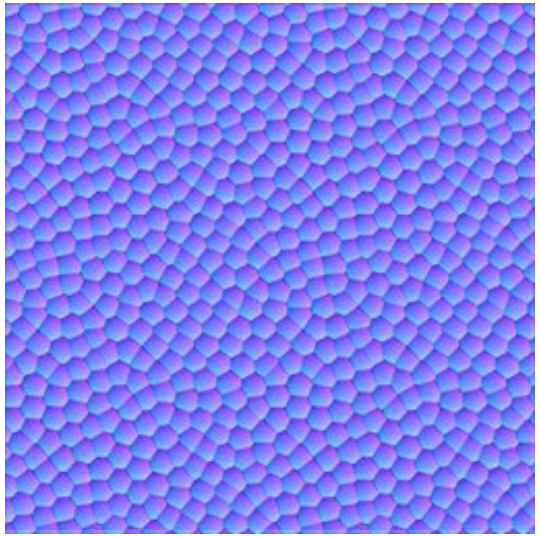
Surface Detail

Bump mapping



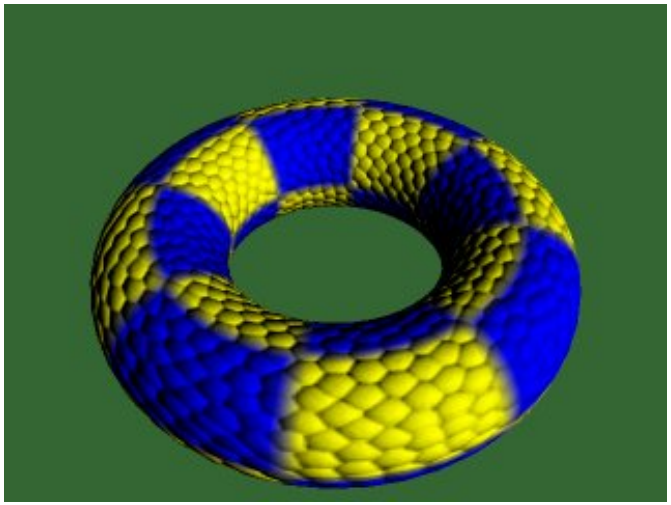
original textured model

+



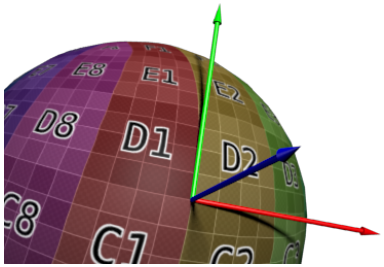
normal map

=



final bump mapped model

A more direct approach is to store n'_s, n'_t and n'_z (in tangent space) directly in RGB channels of the bump map texture. This normal will be the one used. A (0,0,1) in tangent space value will produce the original normal.



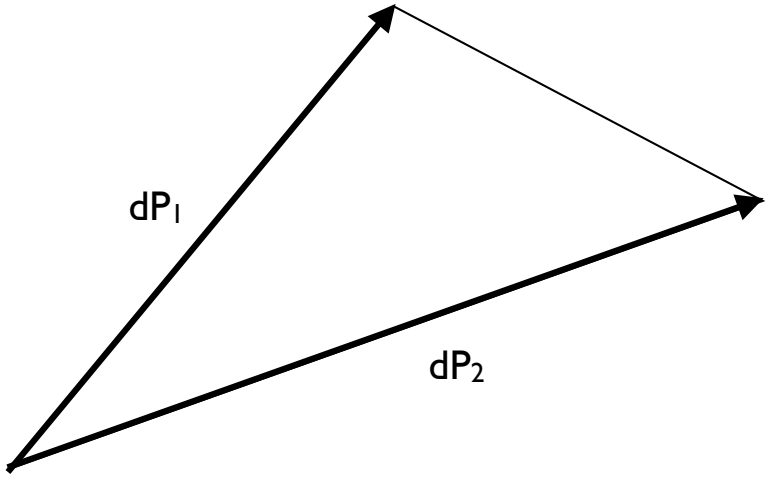
Tangent space

Tangent space is easily computed using the original normal vector and the directions of the texture coordinates. The normal vector coordinates are provided with the mesh, and uv coordinates also.

Surface Detail

Bump mapping (how to get the bi-tangent vectors?)

Edges e_1 and e_2 of a triangle correspond to the changes in positions $d\mathbf{P}_1$ and $d\mathbf{P}2$, and to the changes in texture coordinates (du_1, dv_1) and (du_2, dv_2), respectively.



Assemble the system:

$$d\mathbf{P}_1 = du_1 * \mathbf{T} + dv_1 * \mathbf{B}$$

$$d\mathbf{P}_2 = du_2 * \mathbf{T} + dv_2 * \mathbf{B}$$

Solve for the unknowns **T** and **B**!

T=(T_x,T_y,T_z) and **B**=(B_x,B_y,B_z) can be computed in object coordinates and stored as vertex attributes

Bump mapping (how to get the bi-tangent vectors?)

$$\begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

$$\begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix}$$

$$\begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} N_x \\ N_y \\ N_z \end{bmatrix}$$

Converts from tangent space to model space

$$\begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix}^T \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix}^T \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix}^T \begin{bmatrix} N_x \\ N_y \\ N_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Converts from model space to tangent space*

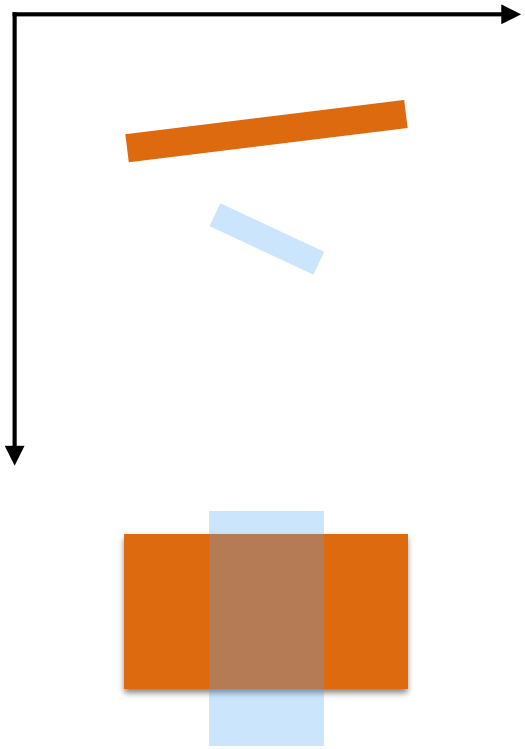
* lighting can be performed in tangent space, if we also convert light and view direction vectors

Transparency

"Glasses" by Gilles Tran (2006), using povray

Transparency

Transparent polygon 1 in front of opaque polygon 2



What if we have more transparent polygons in front?

The methods will have to be applied recursively, in back to front order…

Without Refraction

- Without refraction, light rays pass through transparent objects without being "bent"
- Interpolated transparency:

$$I_\lambda = (1 - k_{t1})I_{\lambda 1} + k_{t1}I_{\lambda 2}$$

Opacity of polygon 1

Transmission coefficient = transparency of polygon 1

- Filtered transparency:

$$I_\lambda = I_{\lambda 1} + k_{t1}O_{t\lambda}I_{\lambda 2}$$

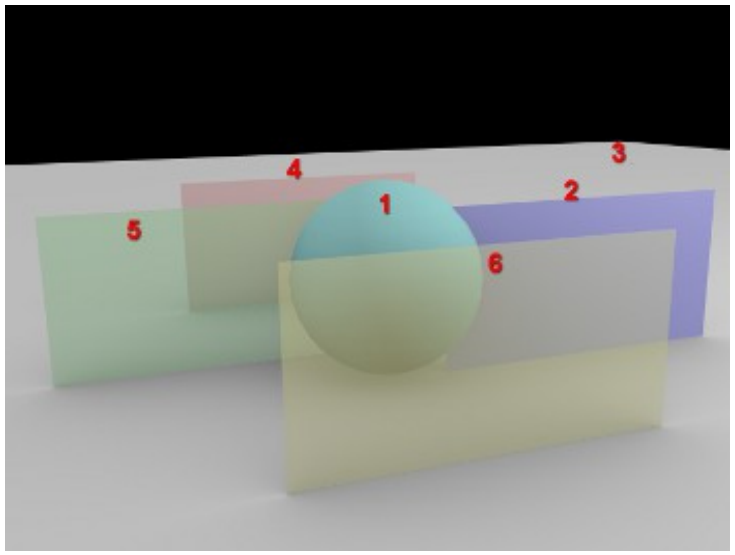
Polygon's transparency colour (Coloured filters will have different values for each wavelength)

Transparency

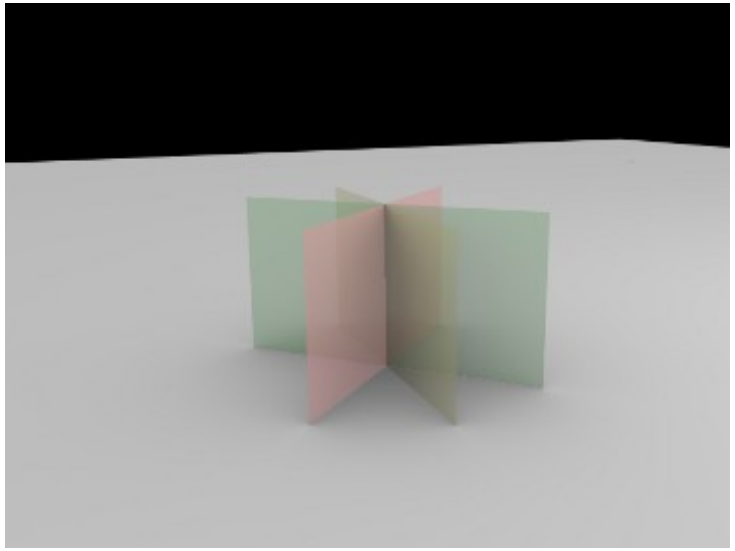
Implementation

1. Draw all opaque objects using traditional z-buffer
2. Create 4 additional buffers with the following, for each pixel: transparency value, flag (initially off), colour and depth (initially closest possible value)
3. Render each transparent object and, for each pixel:
    1. if its z value is closer than opaque z-buffer and more distant than current transparent z-buffer, then set flag and save colour, transparency and transparent z-value.
4. When all objects have been processed, the transparent object buffers have the information about the more distant transparent objects at each pixel (where the flag value is set).
5. Update opaque buffers by blending colour and z-buffer from transparent buffers.
6. Repeat the process to render closer objects while the flag buffer has been modified at least once.

Transparency



- objects 1, 2, 3 and 4 are opaque and rendered first
- object 5 is transparent and needs to be rendered before object 6, also transparent
- Object depth sorting could solve the problem in this example



- In this case all polygons are transparent and there is no object sorting valid for all pixels
- Either split each polygon by its intersection lines with other polygons and perform object sorted rendering or…
- … apply the algorithm presented before.

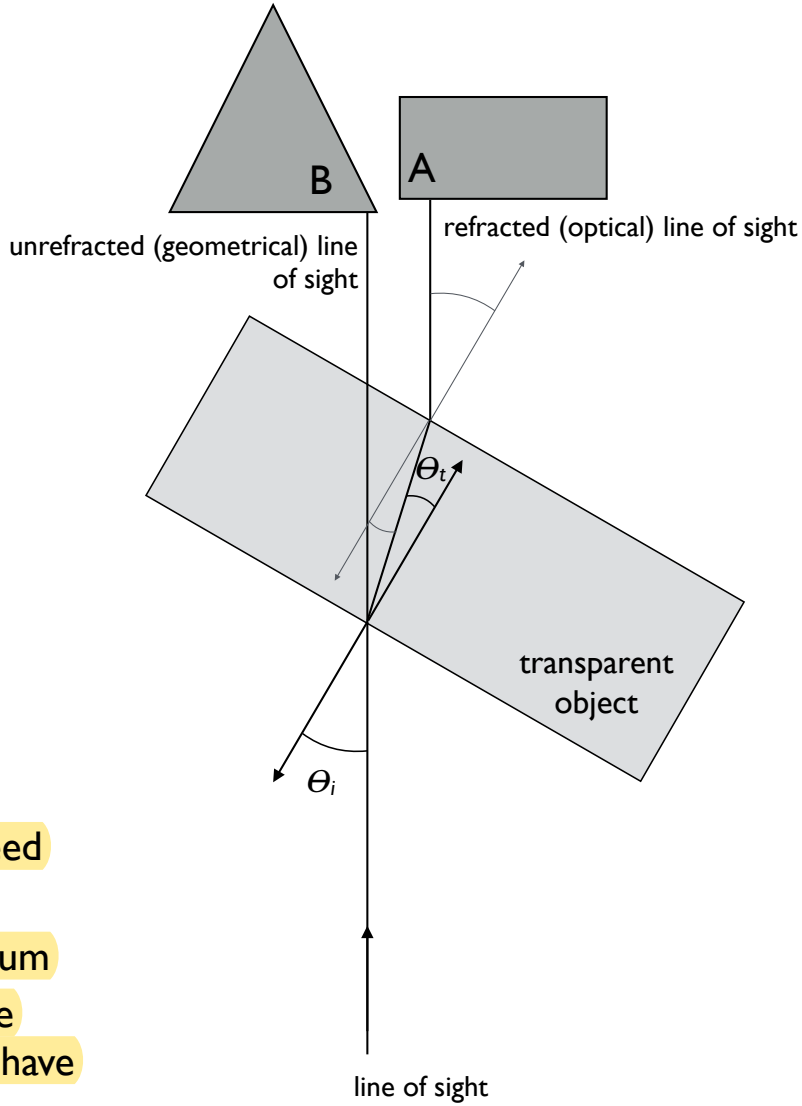
Transparency

Refraction

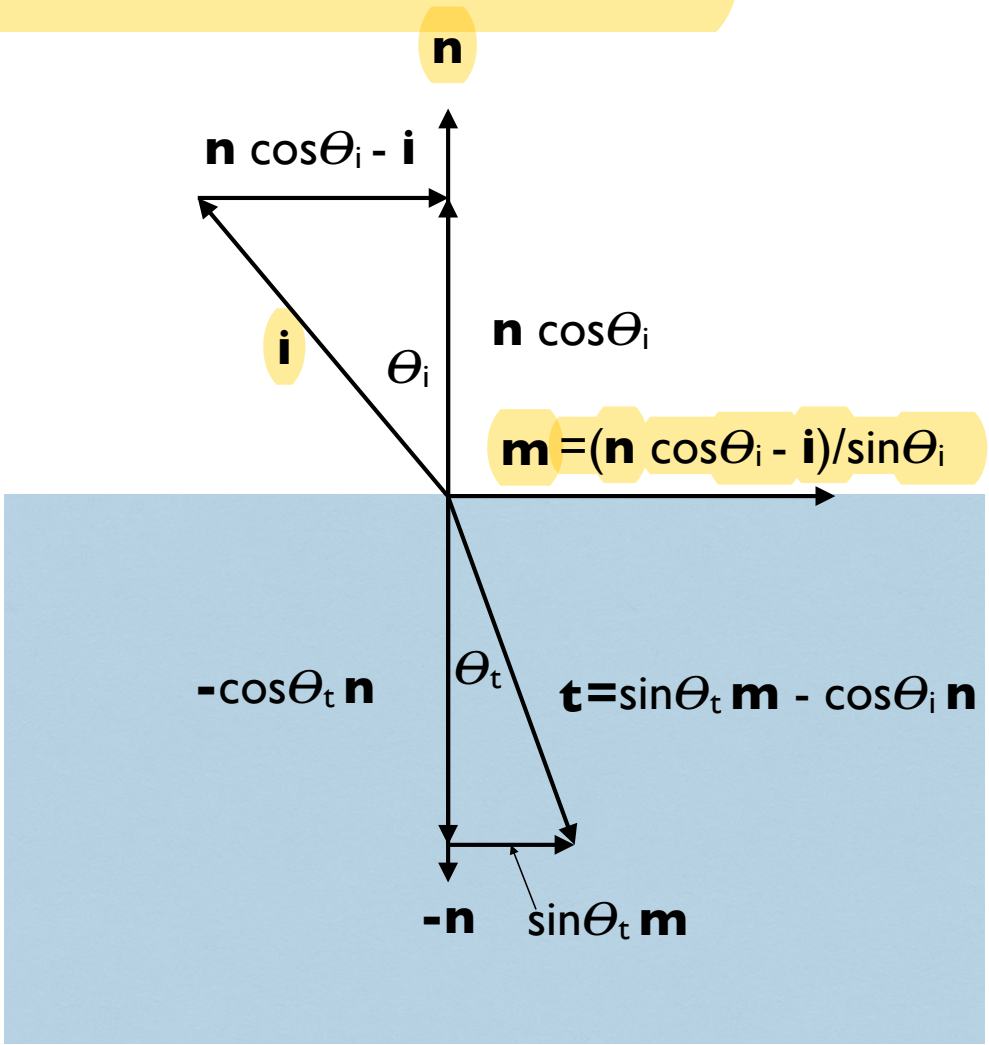
- Without refraction, object B would be visible along the geometrical line of sight
- With refraction, object A will be visible along that line of sight
- Light rays "bend" according to **Snell's law**:

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_{t\lambda}}{\eta_{i\lambda}}$$

η_λ - ratio between speed of light at certain wavelength λ in a vacuum and its speed inside the medium. All materials have values greater than 1.



Refraction Vector



Unit vector along transmitted direction:

$$\mathbf{t} = \sin\theta_t \mathbf{m} - \cos\theta_t \mathbf{n}$$

replacing **m**:

$$\mathbf{t} = \frac{\sin\theta_t}{\sin\theta_i}(\mathbf{n}\cos\theta_i - \mathbf{i}) - \cos\theta_t \mathbf{n}$$

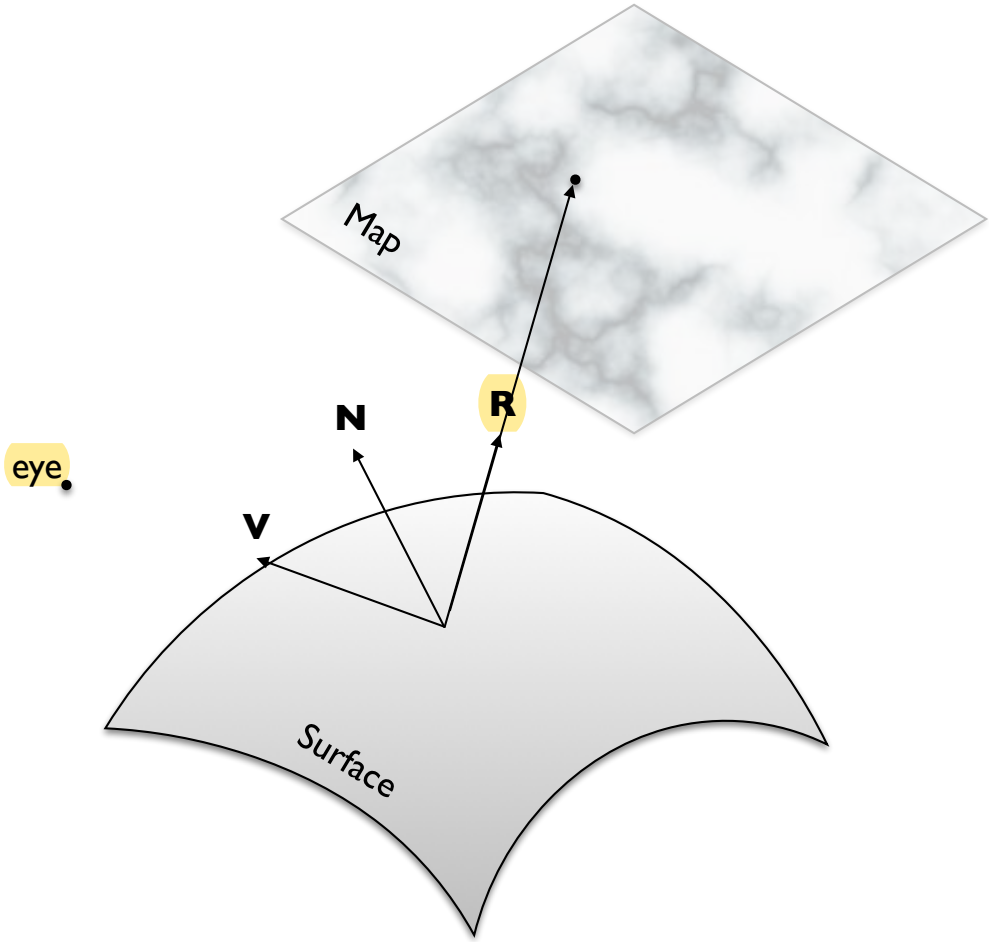
or:

$$\mathbf{t} = \frac{\eta_i}{\eta_t}(\mathbf{n}\cos\theta_i - \mathbf{i}) - \cos\theta_t \mathbf{n}$$

Inter-object Reflections

Reflection Maps

- Inter-object reflections (both specular and diffuse) occur when surfaces reflect other surfaces
- Reflection mapping (or environment mapping) model specular inter-object reflection
- mapping each vertex of a polygon will give the corresponding reflection polygon in the map
- The way R is used to index the map depends on the particular method.



Inter-object Reflections

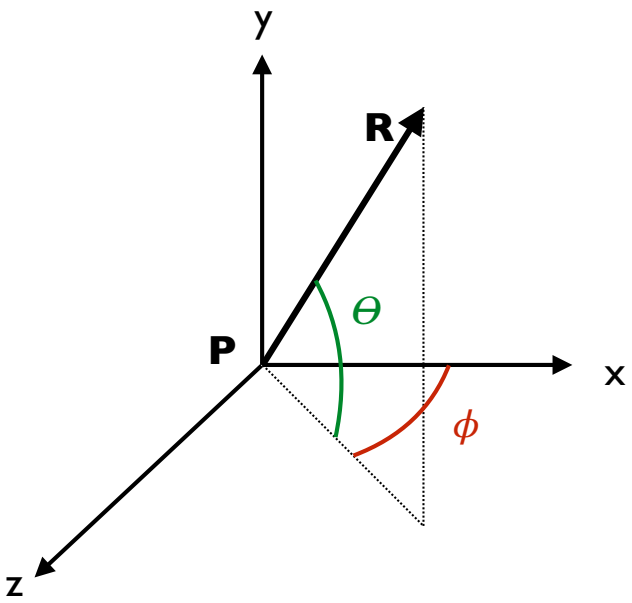
Reflection Maps: spherical mapping

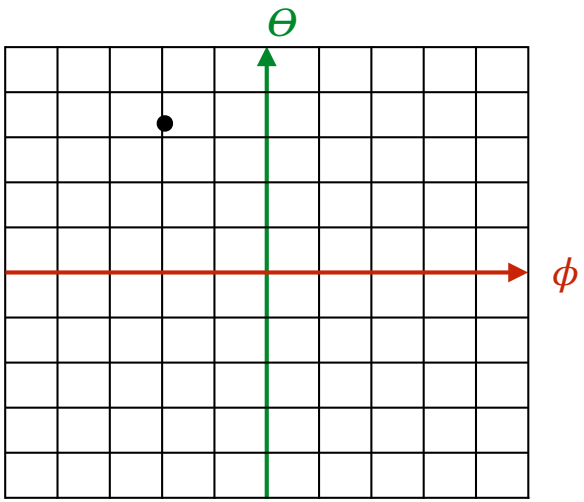
- Spherical environment mapping starts by choosing a centre of projection to map the environment to be reflected onto a sphere surrounding the objects to be rendered

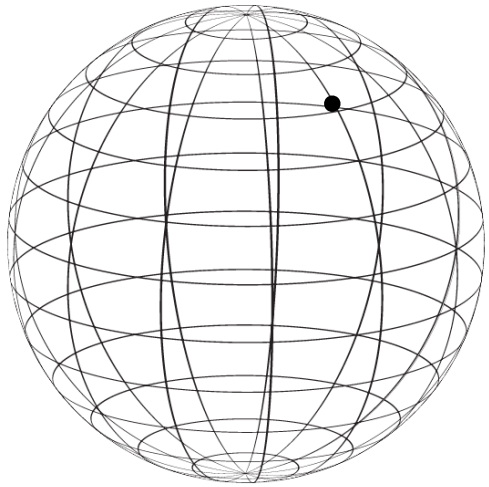
- The environment can be treated as a 2D texture
- On each point of the object to be rendered, the view vector **V** is reflected about **N** to give **R**.
- **R** is used to index the environment map by using polar coordinates
- The environment mapped sphere is assumed to be infinitely big
  - problems at the caps of the sphere
  - Distortion
  - Not valid for concave objects (no self reflections are visible)

Inter-object Reflections

Reflection Maps: spherical mapping







- The reflection vector (x,y,z) coordinates are mapped to polar coordinates (Θ,ϕ).
- Latitude (Θ) and longitude (ϕ) are used to index the sphere map

Inter-object Reflections

Reflection Maps: cube mapping

- Cube mapping is preferable to spherical mapping because:

✓ causes less distortion

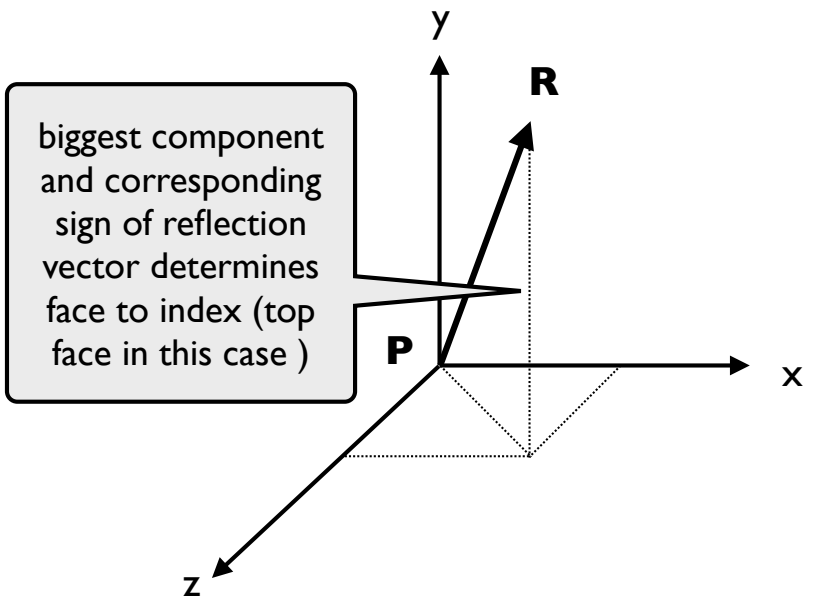
✓ texture resolution is less critical (each face has the same resolution as the whole map of spherical mapping)

- A map is generated for each face of the cube by projecting the scene along the direction of that face
- Using the normalised reflection vector **R**, the component with the highest magnitude tells which pair of opposite faces to look for. The sign chooses between the pair.

Inter-object Reflections

Cube mapping

Cube map



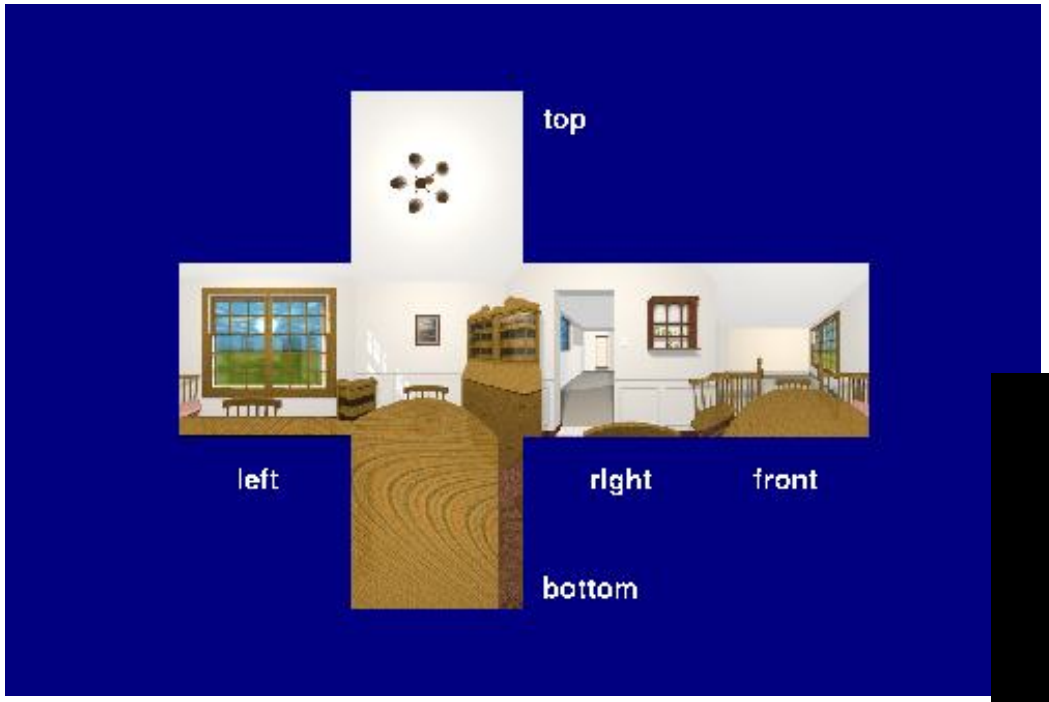
biggest component and corresponding sign of reflection vector determines face to index (top face in this case)

viewpoint to cube map generation

Cube map mapped to an object (in this case also a cube)

Inter-object Reflections

Cube mapping

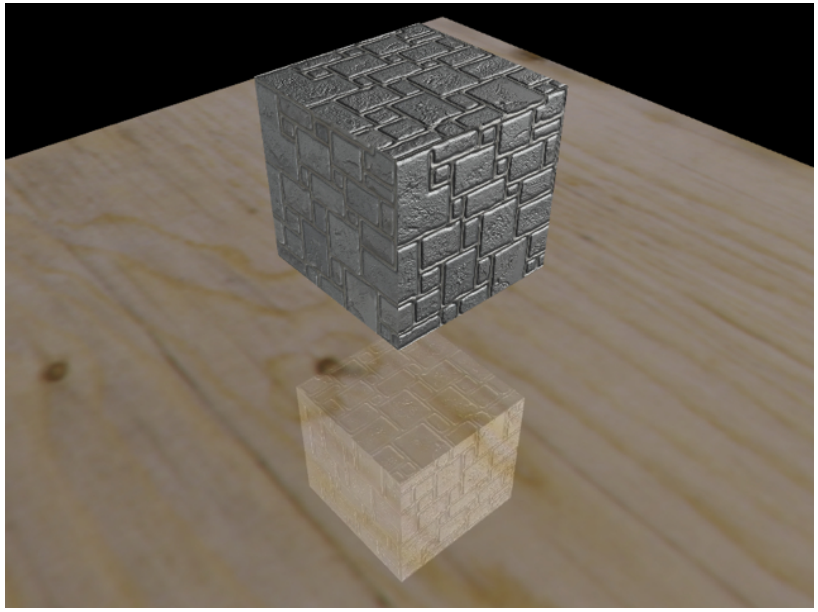


What if the objects in the scene are moving or if new objects are being added?

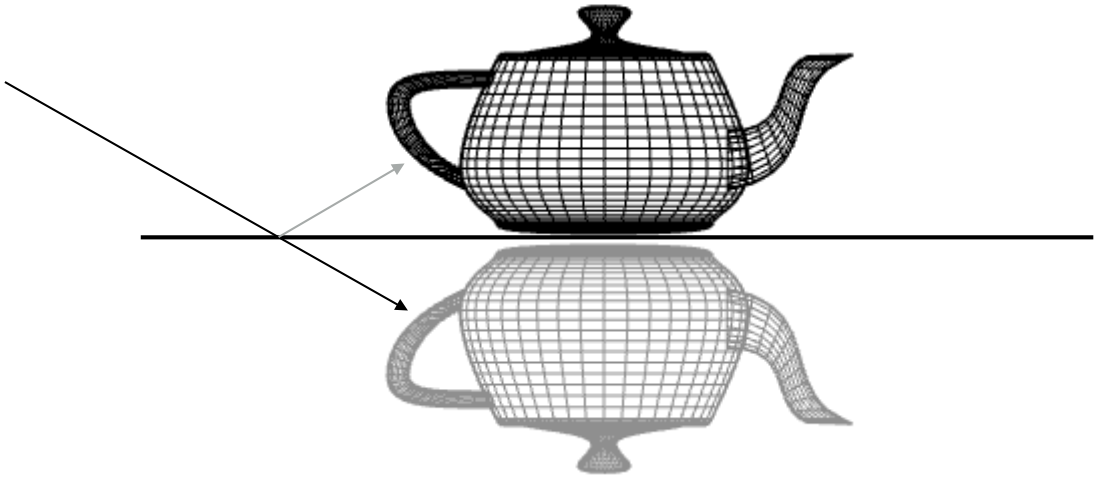
Inter-object Reflections

Reflections on a plane surface

- Reflections on large planar surfaces are better handled by rendering an inverted (mirrored) version of the object

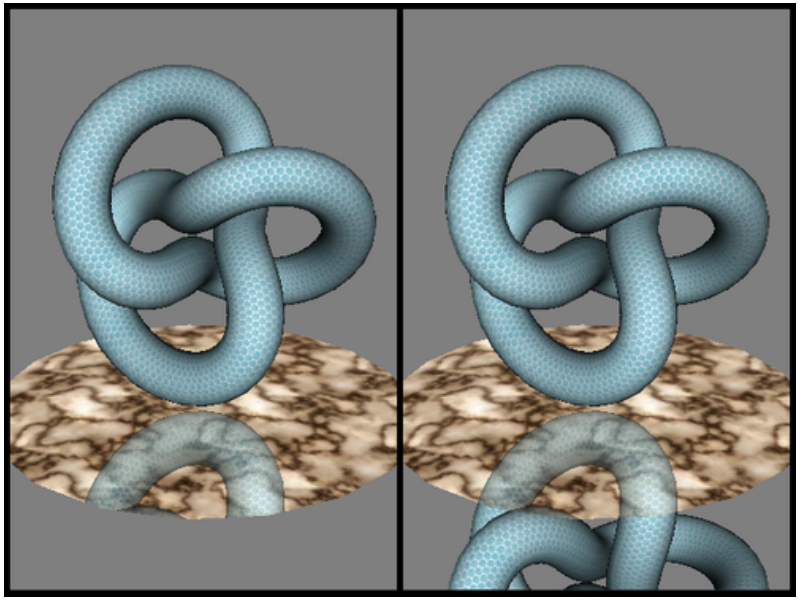


1. Draw inverted model with z-buffer active
2. Draw mirror plane with z buffer active and blend it with the frame buffer contents
3. Draw normal model with z-buffer and no blending



Reflections on a plane surface

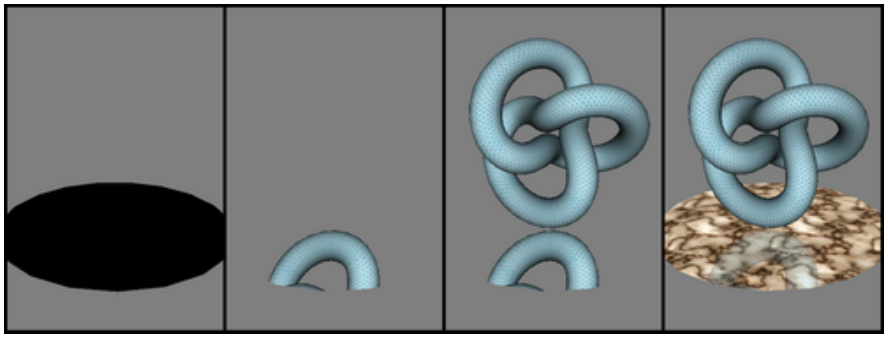
Problem: What if the mirror plane is not big enough?



Right Wrong

Solution:

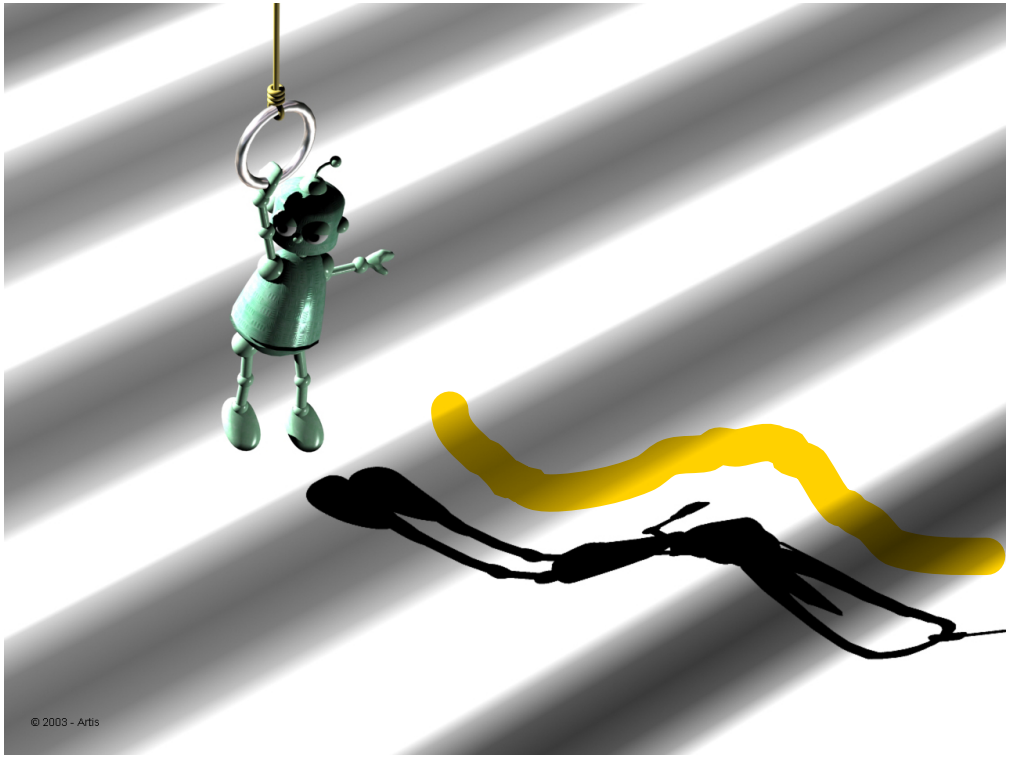
1. Render the mirror plane into a stencil buffer only
2. Render the mirrored version with stencil enabled
3. Clear depth buffer and render actual model
4. Render the mirror plane using front-to-back blending



1 2 3 4

Shadows

Where is the character?



Shadows improve our perception of relative positions

Shadows

- Shadows are relatively easy to include in shading, for point light sources
- Small change to Phong's illumination model:

$$\mathbf{I} = \mathbf{I}_a K_a \mathbf{O}_d + \sum_{l=1}^{L} S_i f_{att,l} \mathbf{I}_{p,l} [k_d \mathbf{O}_d (\mathbf{N} \cdot \mathbf{L}) + K_s \mathbf{O}_s (\mathbf{N} \cdot \mathbf{H})^n]$$

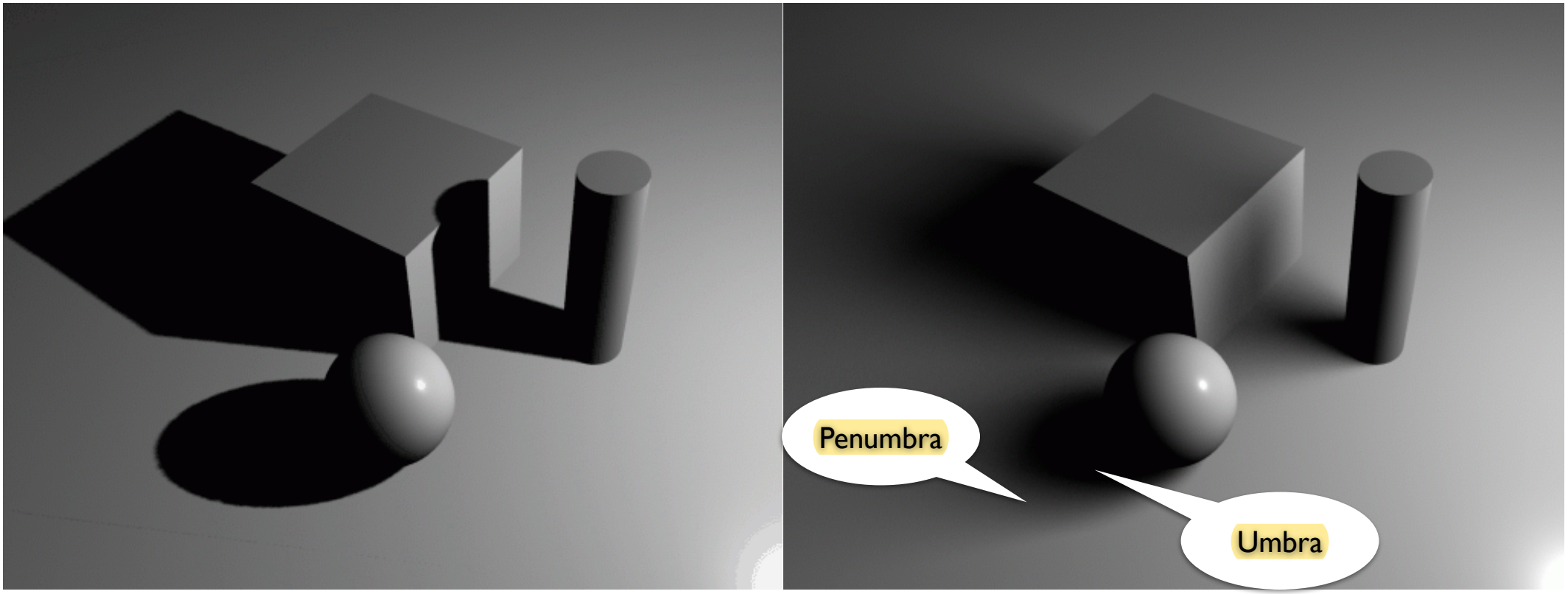
$$S_i = \begin{cases} 0, \text{ if light } i \text{ is blocked} \\ 1, \text{ if light } i \text{ is not blocked} \end{cases}$$

- All or nothing doesn't take into account indirect illumination (light reflected off other surfaces)
- For area light sources, shadow intensity depends on the amount of light that is blocked…

Shadows

Point light sources

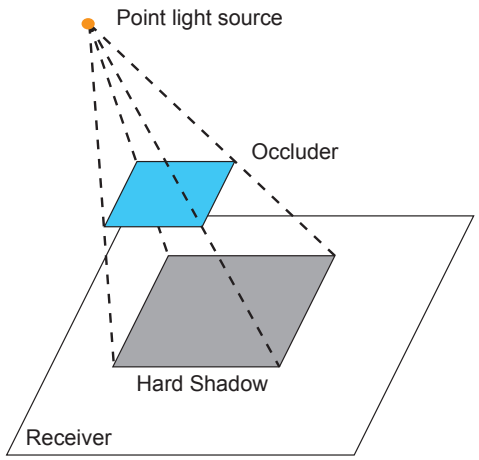
Area light sources



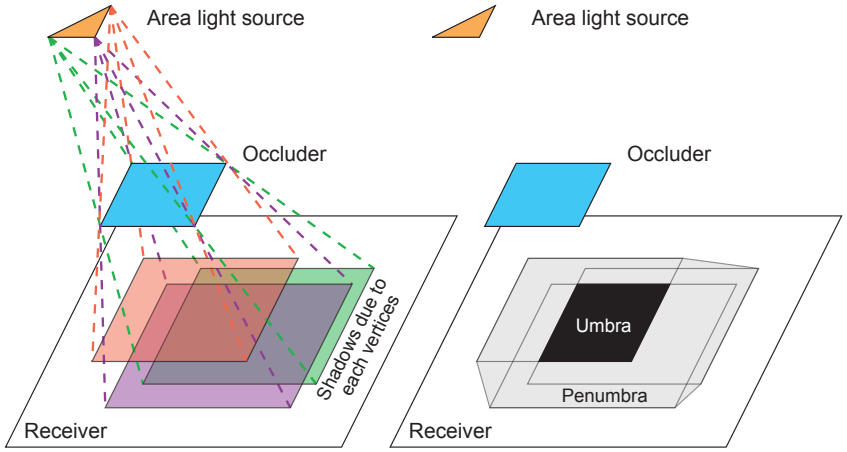
hard shadows

soft shadows

Hard and Soft shadows



(a) Geometry of hard shadows



(b) Geometry of soft shadows

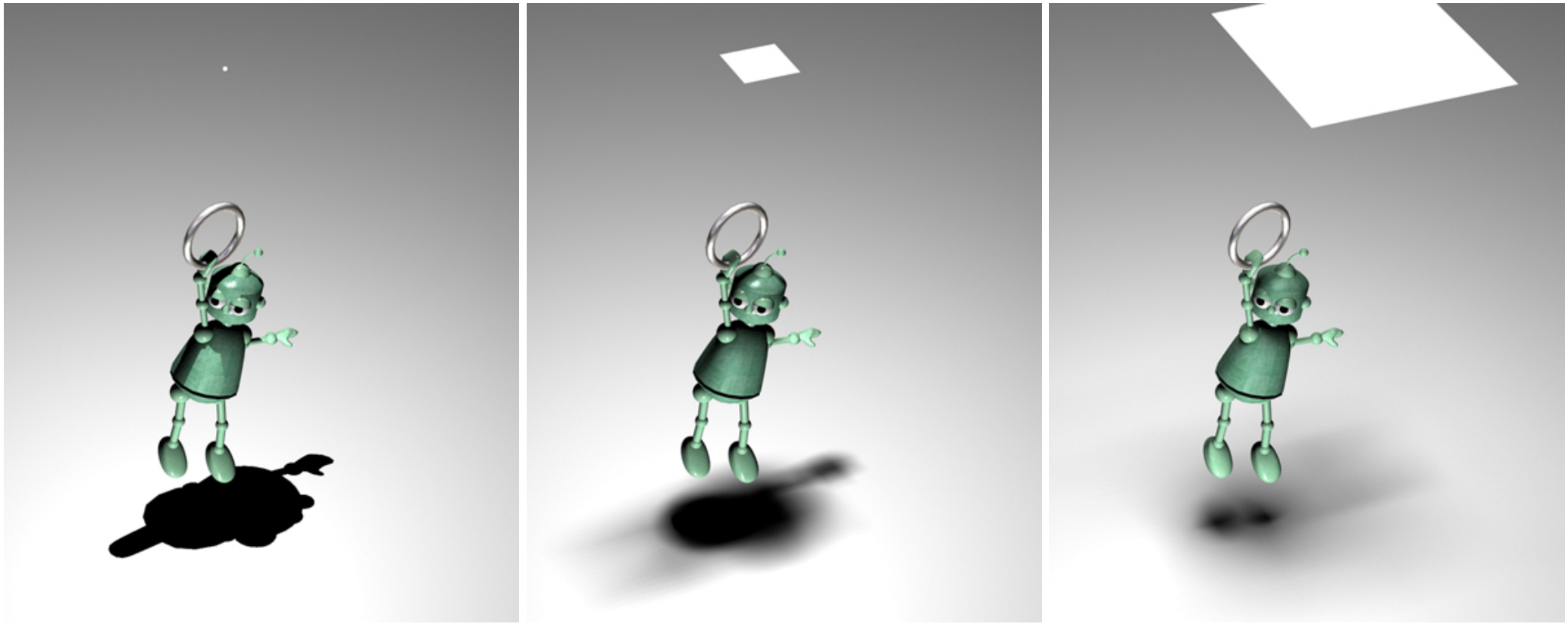
(c) Illustration of hard shadows

(d) Illustration of soft shadows

Picture taken from: http://maverick.inria.fr/Publications/2003/HLHS03a/SurveyRTSoftShadows.pdf

Shadows

Variation of shadow with light source geometry



Shadows

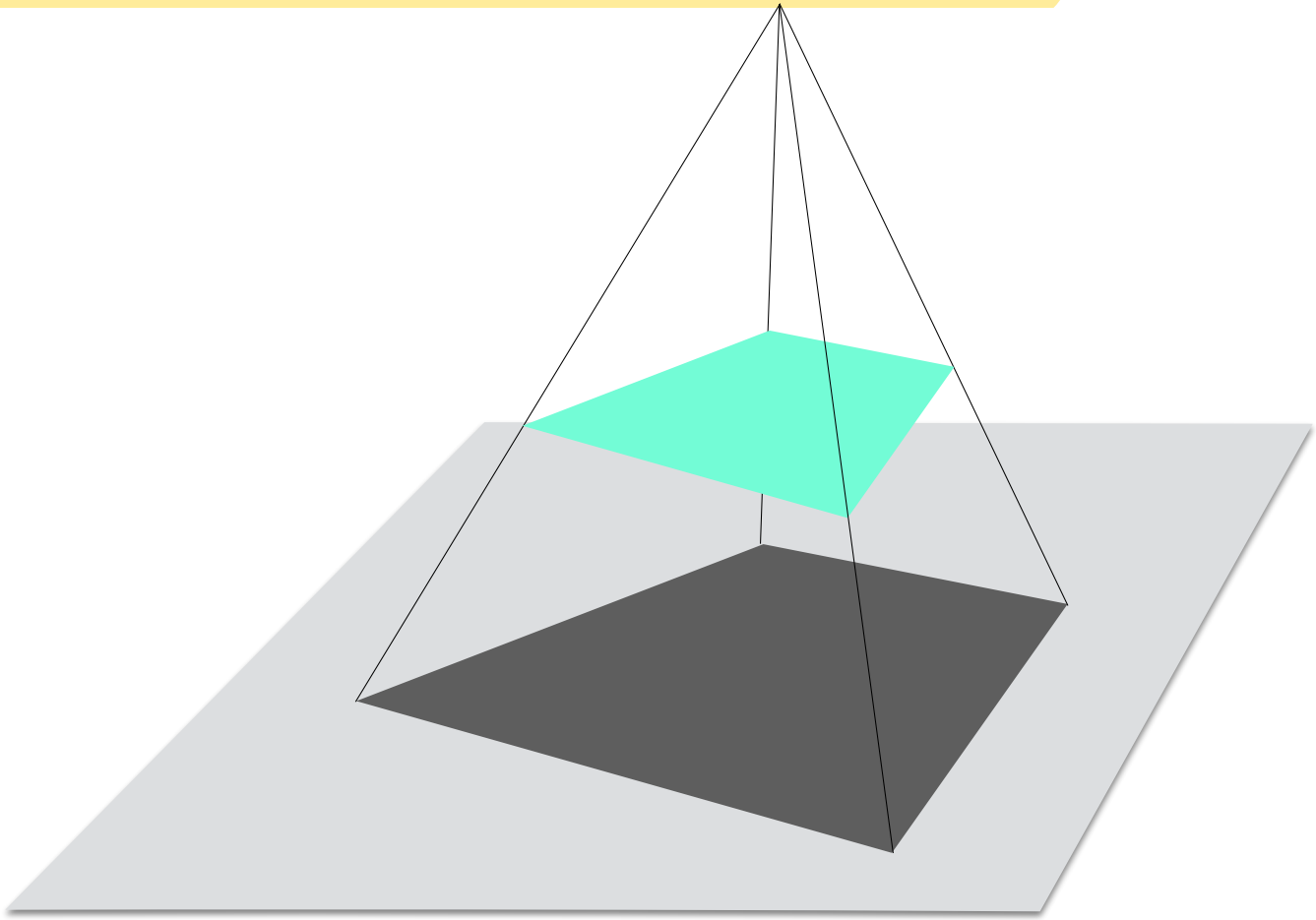
Soft shadows by accumulation of hard shadows

4 samples

1024 samples

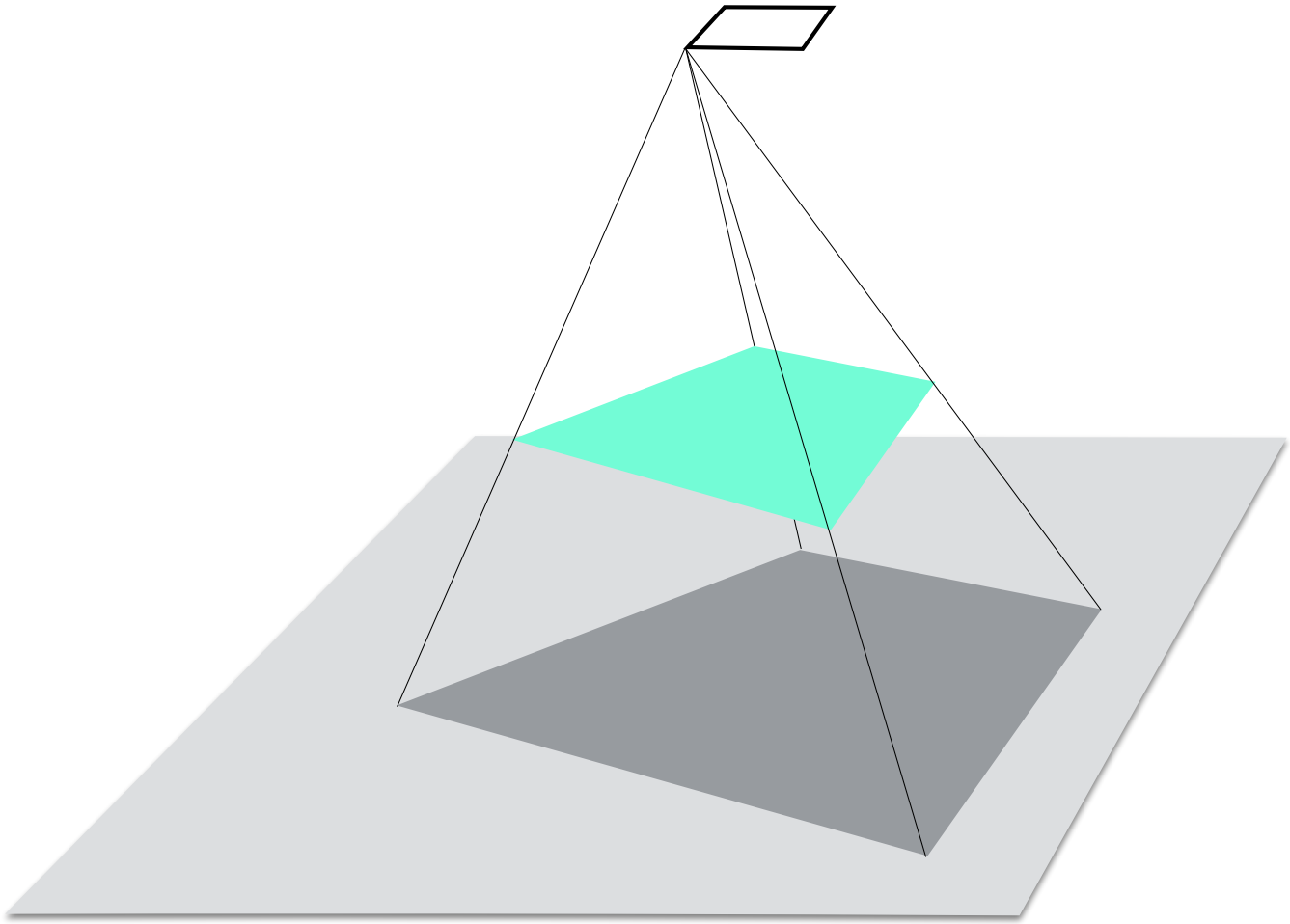
Shadows

A hard shadow projected on a surface



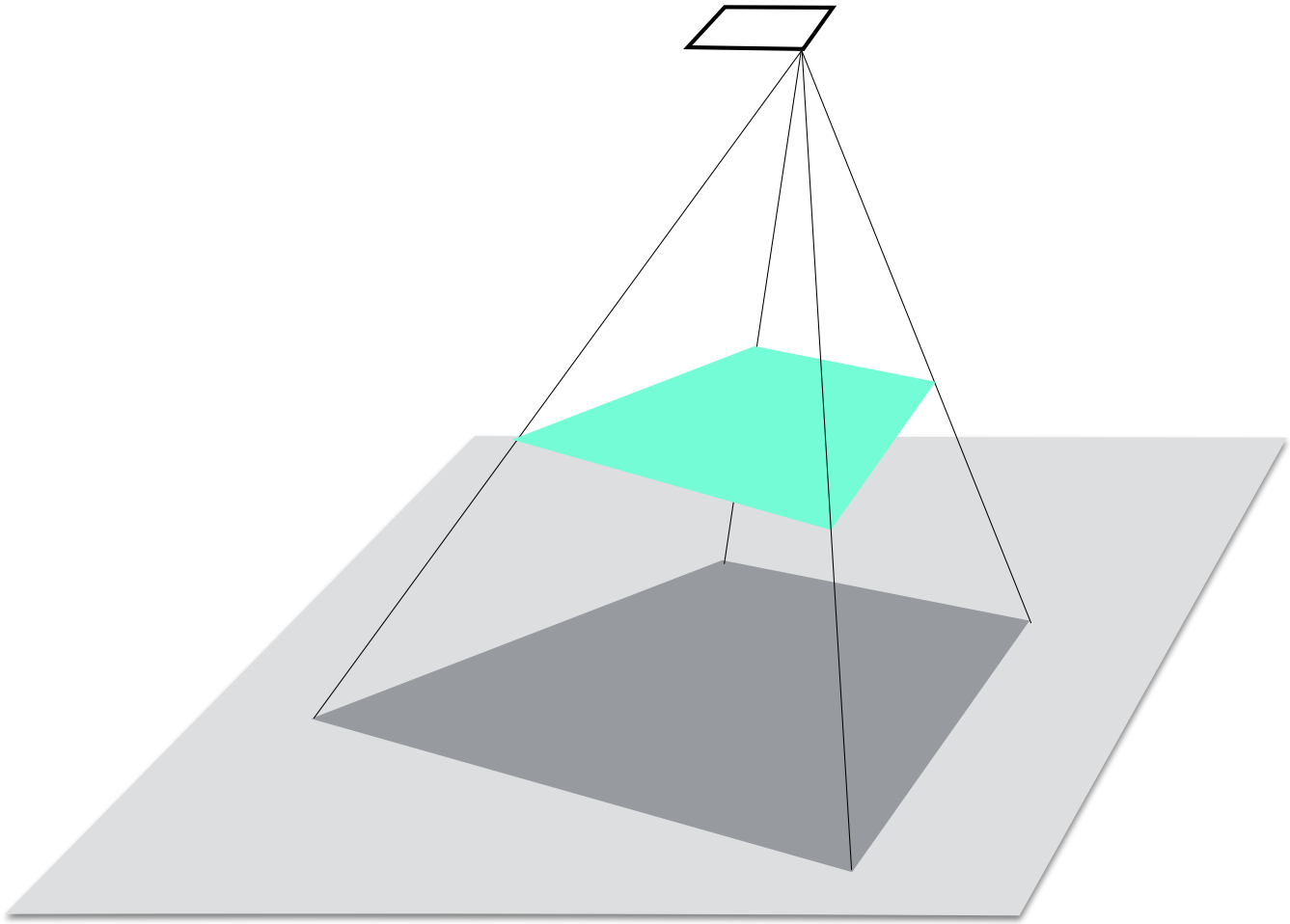
Shadows

Hard shadow produced from point l



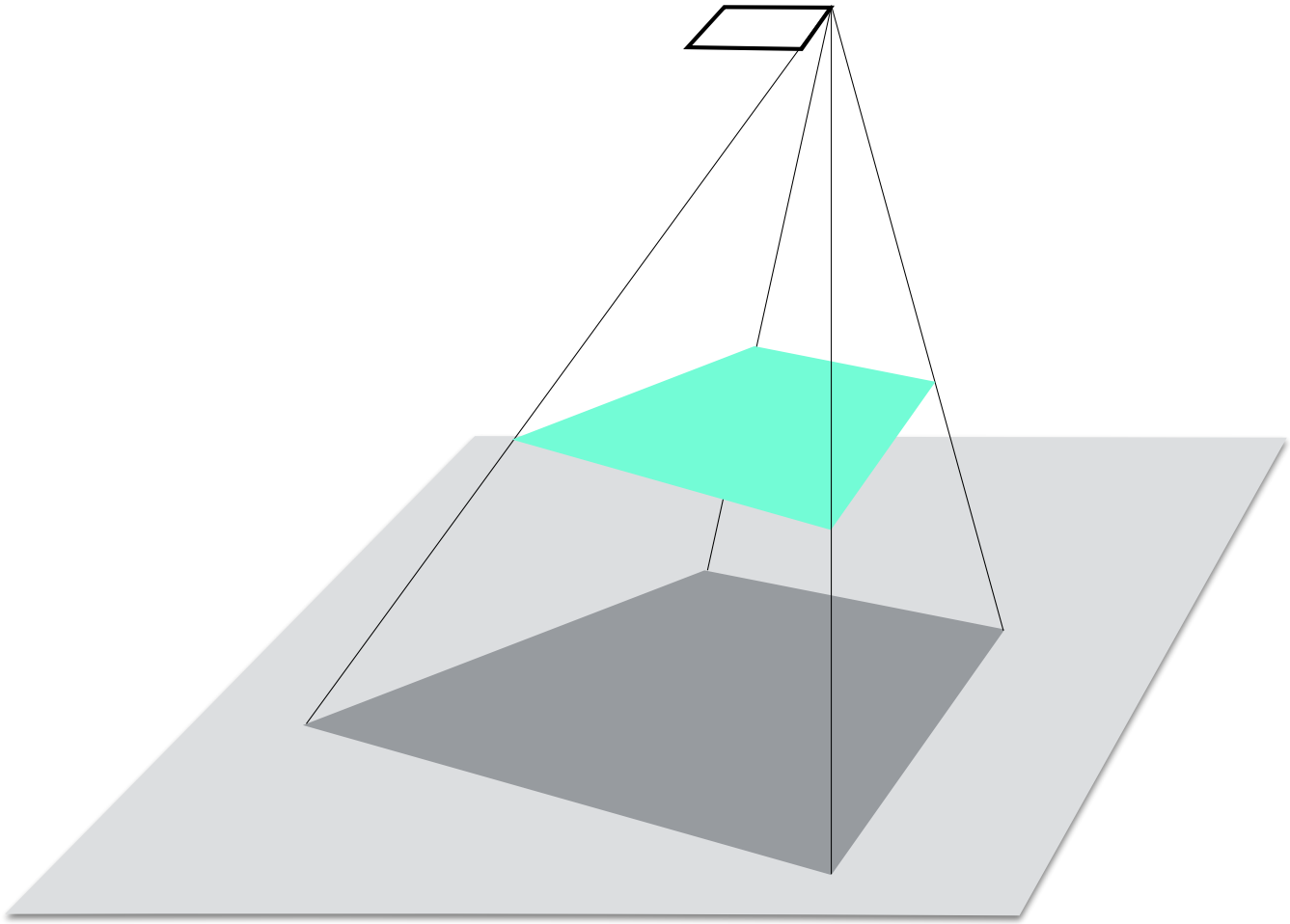
Shadows

Hard shadow produced from point 2



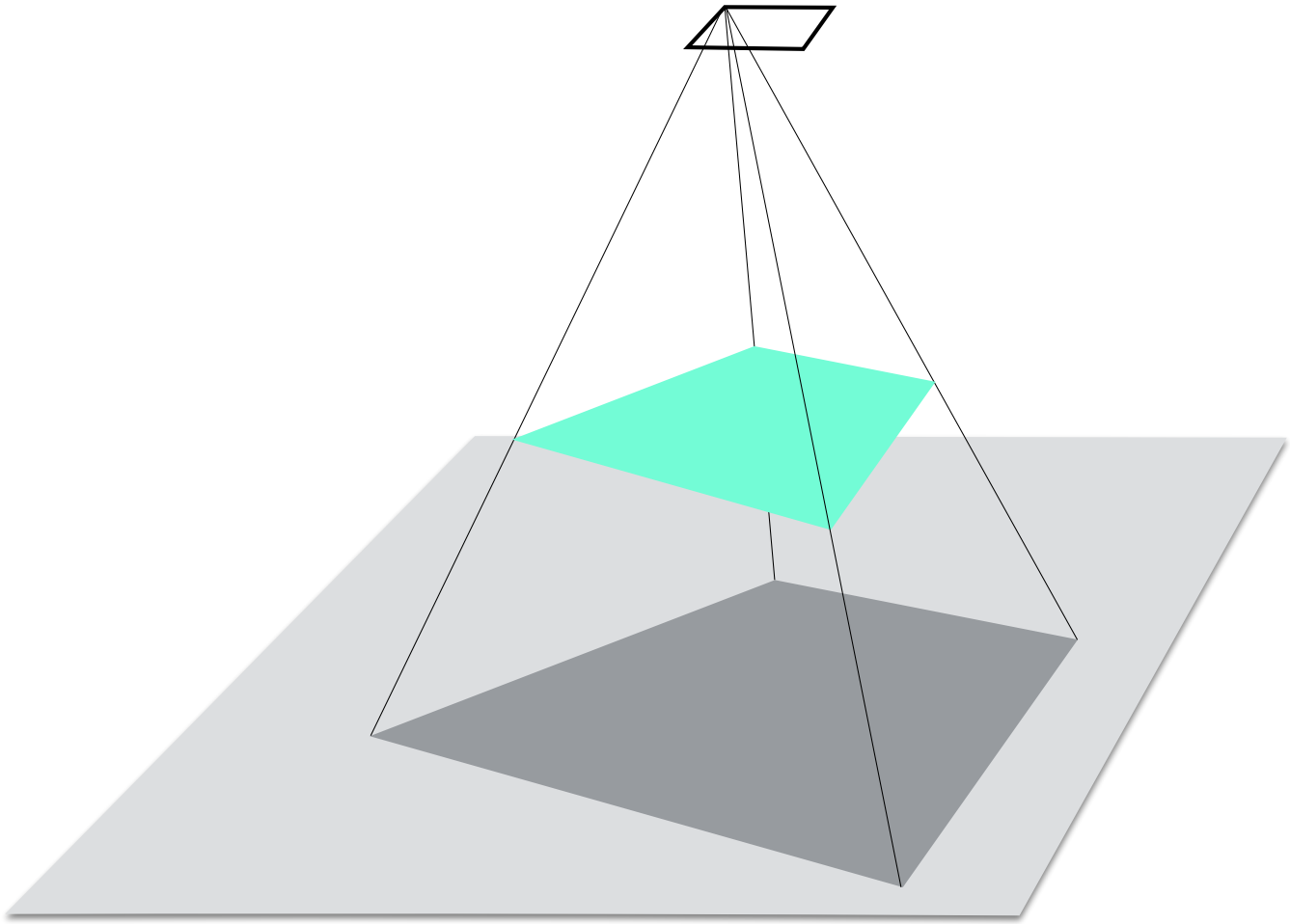
Shadows

Hard shadow produced from point 3

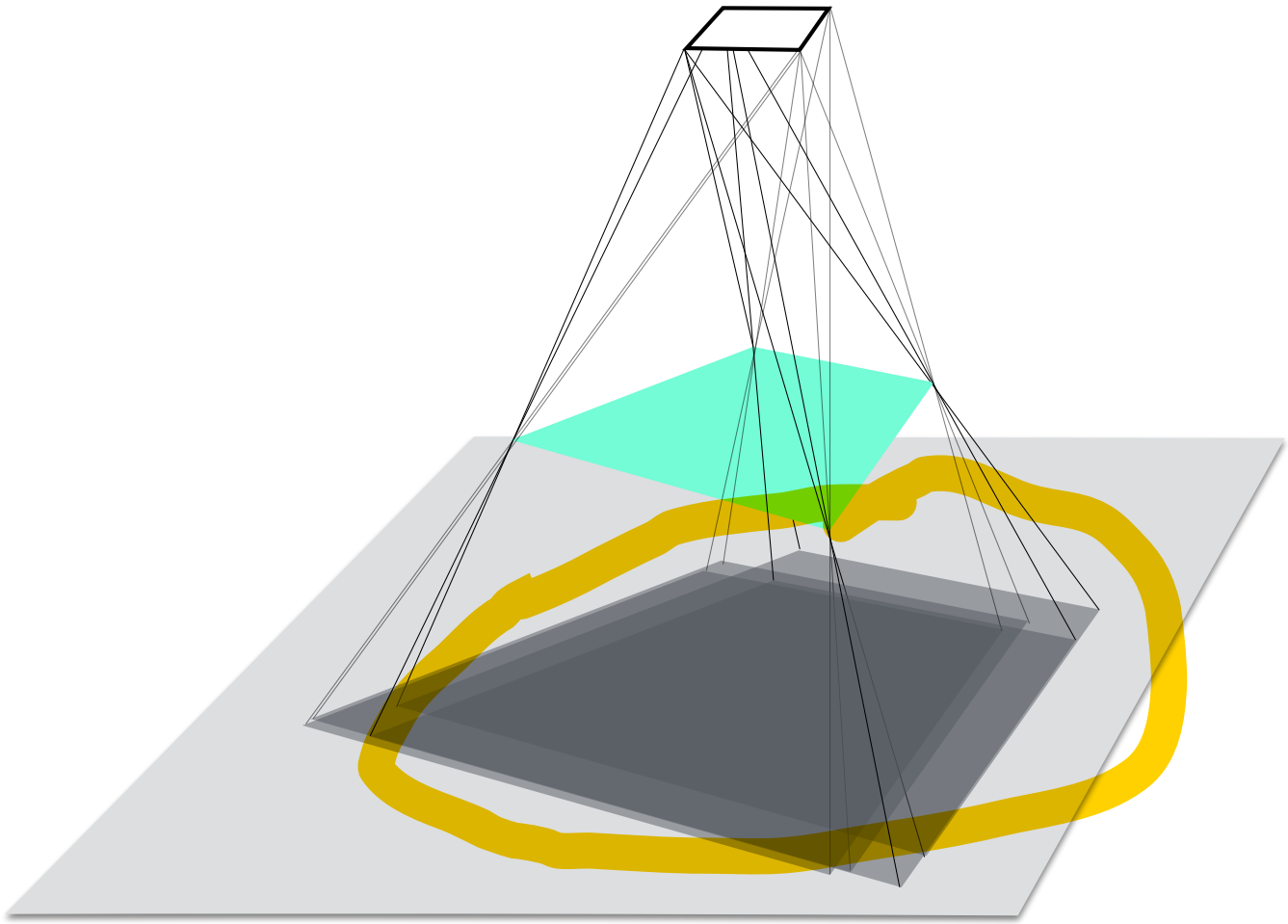


Shadows

Hard shadow produced from point 4



Soft shadow produced by 4 accumulated hard shadows



Shadow generation techniques

- 1968-1970: Integration of shadow polygons with a scan-line hidden surface removal algorithm. Shadows are computed while producing the image
- 1978: Shadow generation based on transformations and clipping. Shadows are precomputed and stored in a database
- 1977: 2 Stage generation with shadow volumes. A shadow volume is generated for each light source and accessed during rendering.
- Shadow determination using z-buffer
- Shadow determination with recursive ray-tracing
- Shadow determination with radiosity method

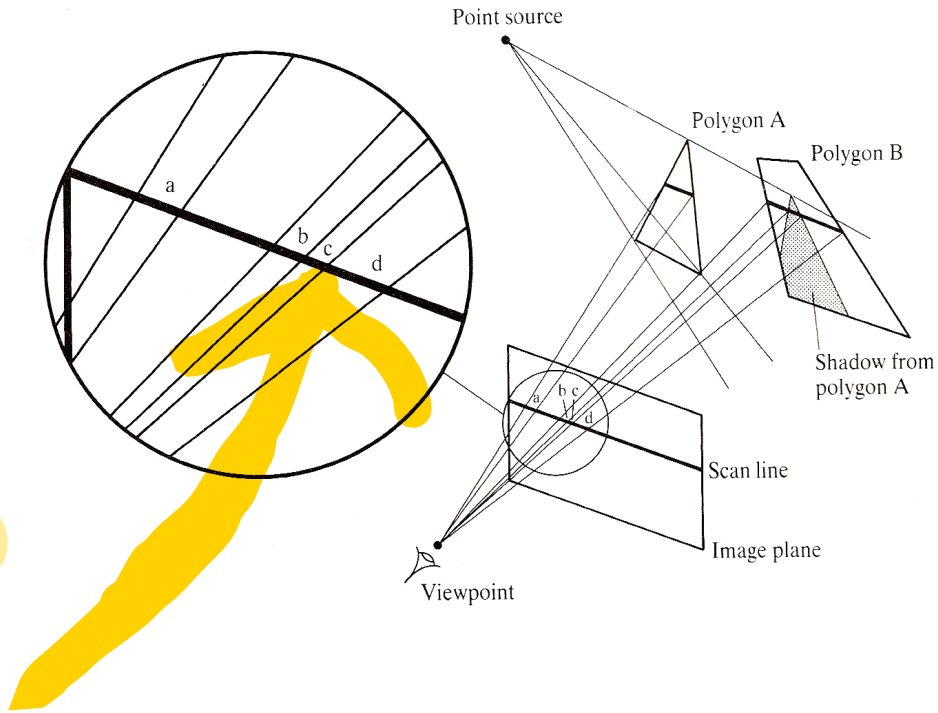
Shadows in scan-line algorithms (68-70)

- In a preprocessing step determine for each polygon **all the other polygons** that can possibly shadow it (shadow polygons). This can be done by surrounding the light with a sphere and projecting all polygons onto its surface.
- During scan-line hidden surface removal, for a polygon, if no shadow polygons exist do as normal, otherwise:

  1. shadow polygon completely overlaps current scan-line segment ⇒ reduce intensity

  2. shadow polygon does not overlap the current scan-line segment ⇒ shade as normal

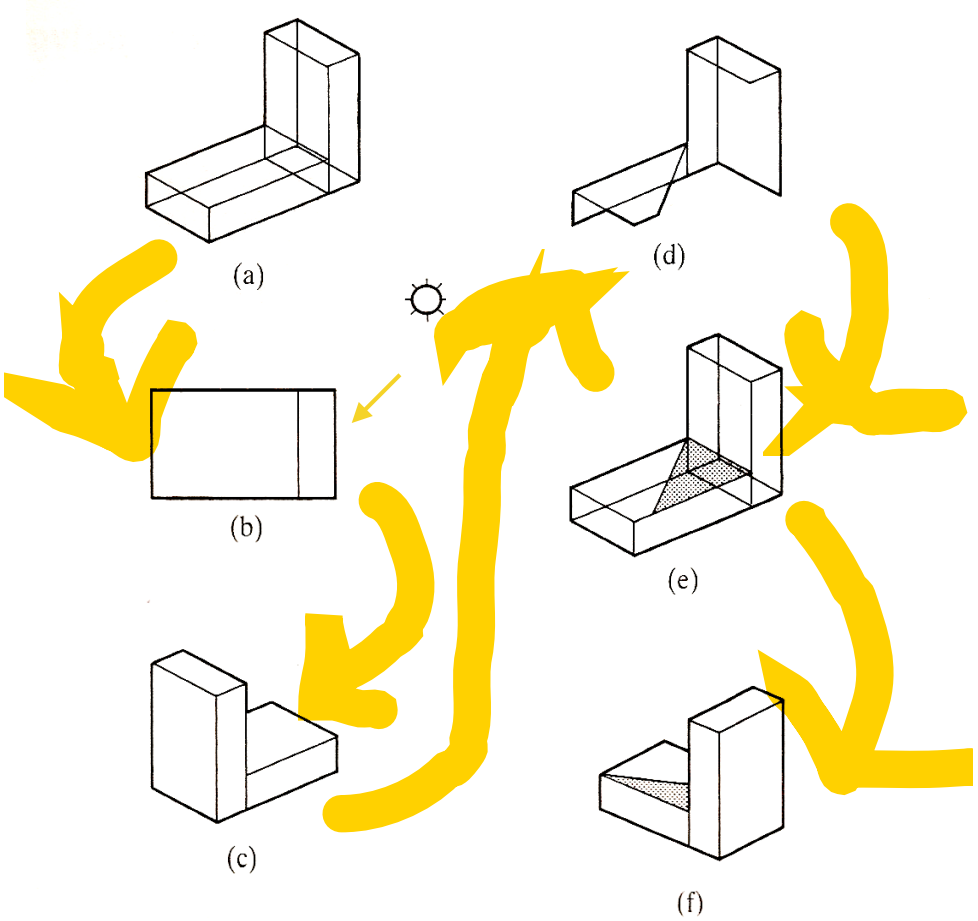
  3. shadow polygon partially overlaps current scan-line segment ⇒ subdivide segment where scan-line segment intersects shadow polygon and restart the step



- Limited to objects modelled with polygons
- Limited to point light sources (hard shadows only)
- Preprocessing stage can take a lot of time

Shadows with transformations and clipping (78)

- A Hidden surface removal algorithm (Weiler-Atherton) is applied to a view computed from the light source to find complete surface detail shadow polygons (c-d) by clipping the original surfaces against the lighted parts.
- Shadow polygons from previous step (d) are transformed back to original object coordinates and a new view independent model is created (e).
- Hidden surface removal computed from the viewpoint is applied to the previous model (f)

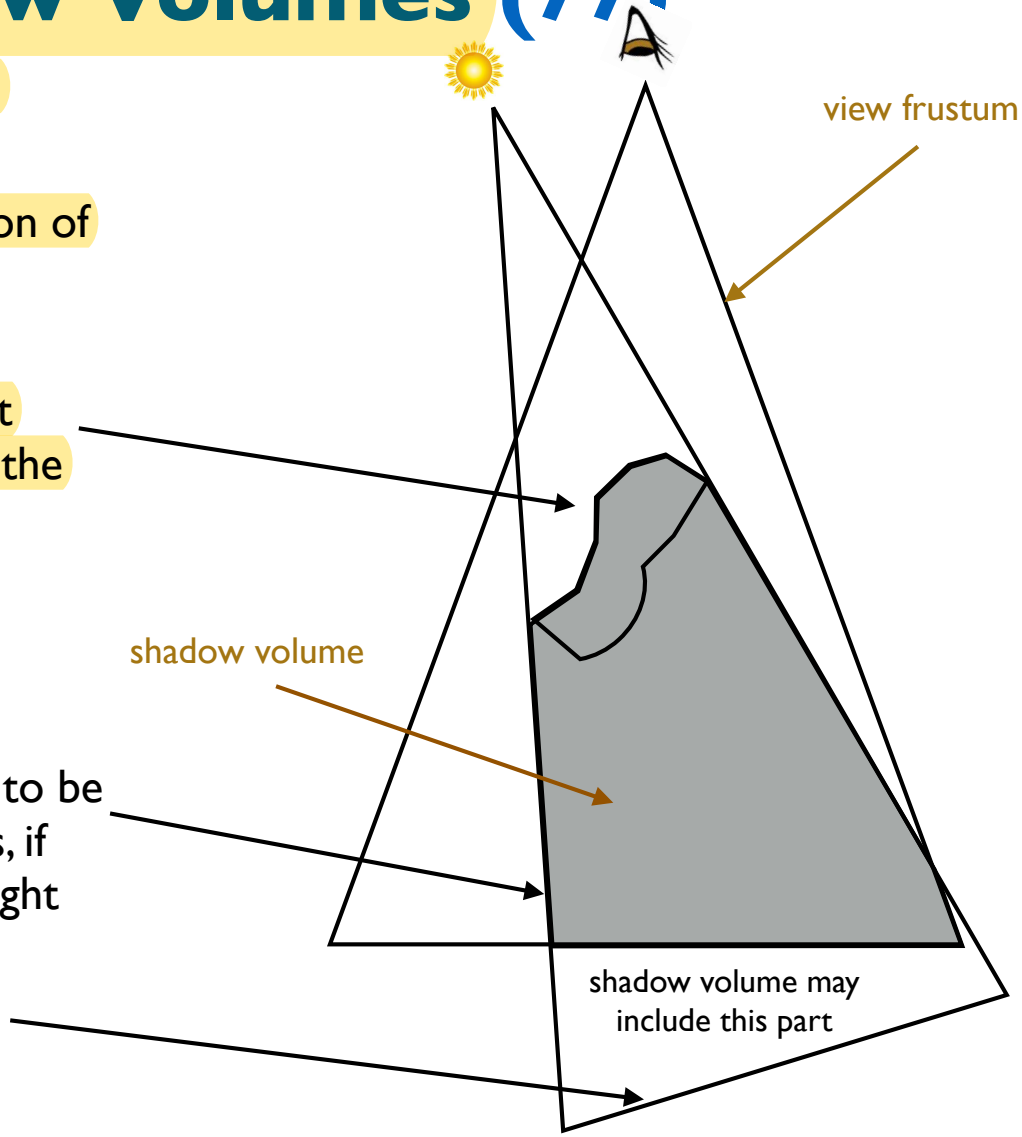


- Limited to objects modelled with polygons
- hard shadows only

+ Computation is performed with object resolution (exact)
+ Illuminated/Shadowed areas can be computed (CAD)

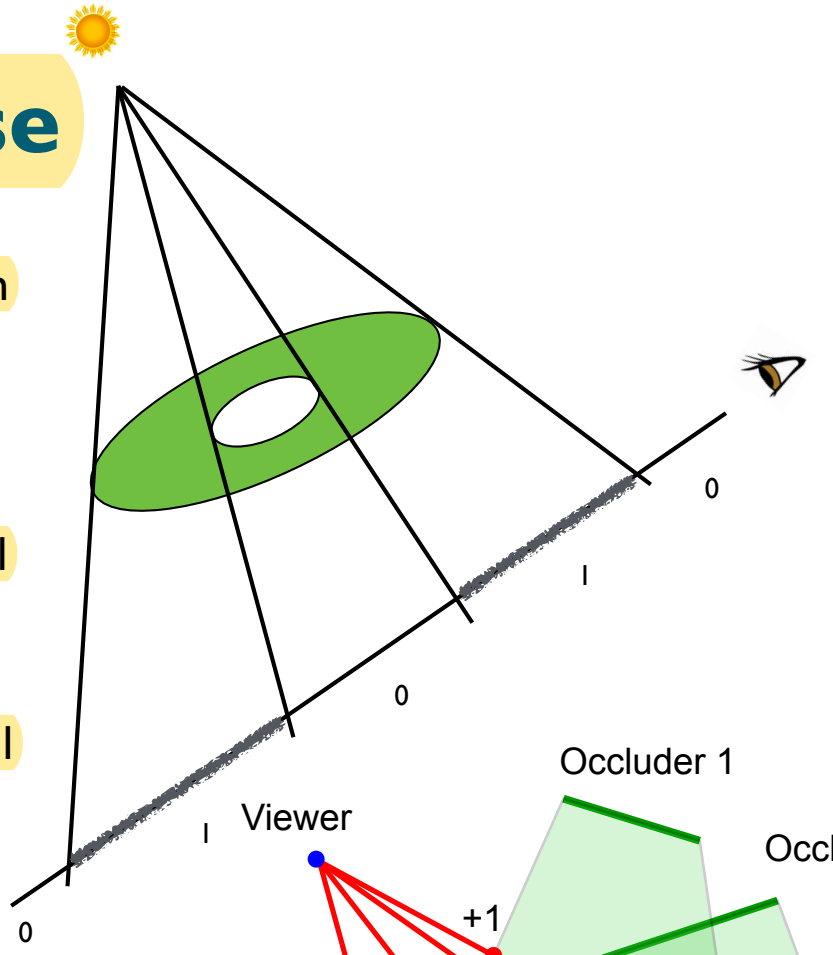
Shadows with Shadow Volumes (77)

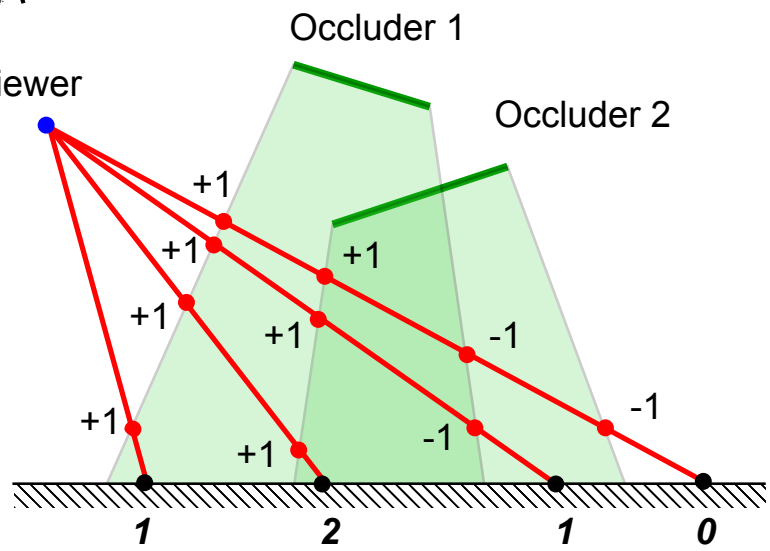
- Shadow volumes are defined for polygonal objects that may cast shadows
- The shadow volume determines the portion of space in the shadow of an object for a particular point light source
- The shadow volume is defined by the front facing polygons of the object viewed from the light source…
- … quadrilaterals with one edge being a silhouette edge of the object…
- … 2 edges connecting the vertices of the silhouette edge to points far away enough to be outside the view frustum. These two edges, if prolonged, would pass through the point light
- … and a remaining edge outside the view volume



Shadow Volumes in use

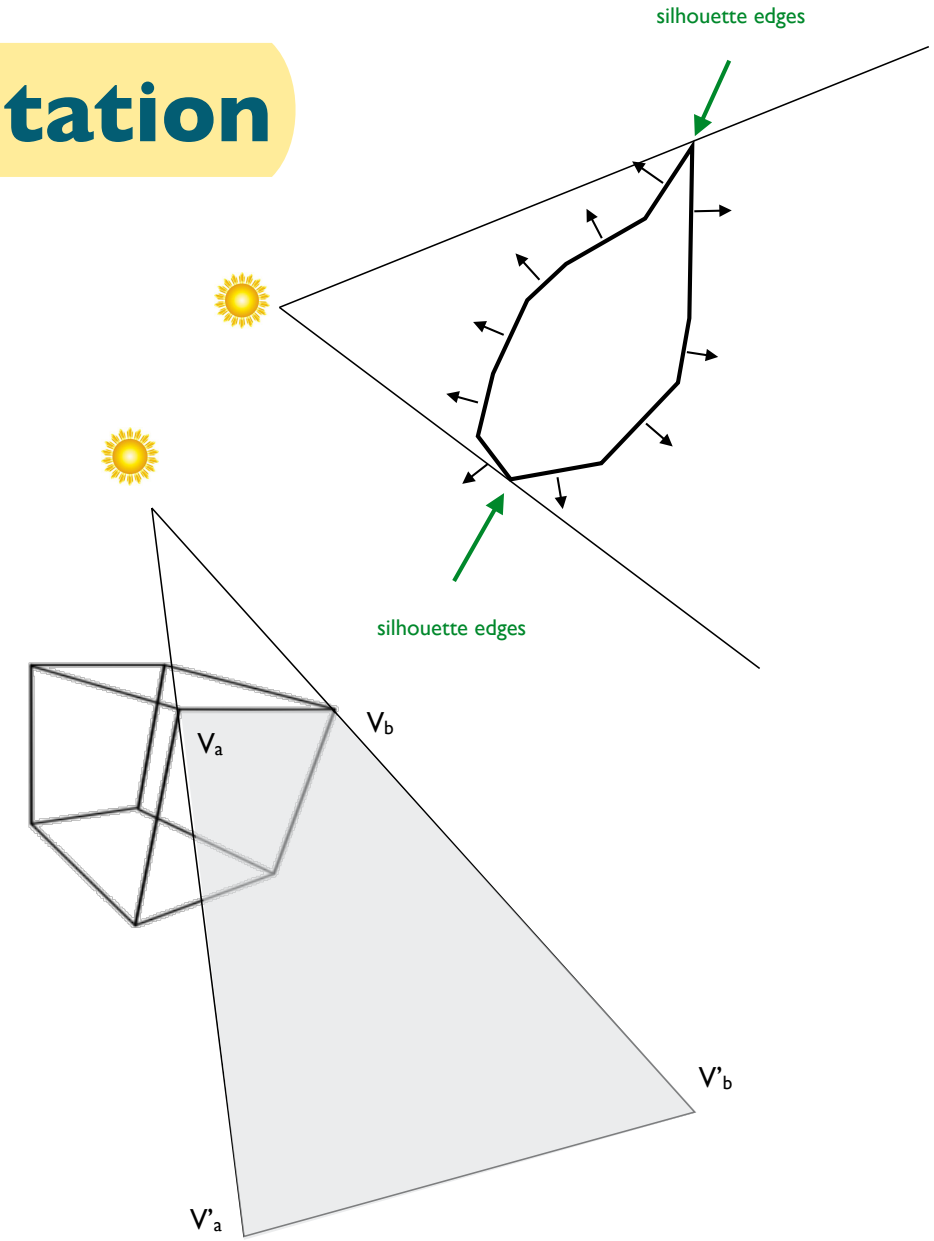
- The basic idea is that if we sort the shadow volume polygons in increasing order of depth from the viewpoint…
- … and distinguish between front facing and back facing shadow polygons…
- …crossing a front facing shadow polygon will increase the shadow counter (entering the shadow volume)…
- … crossing a back facing shadow polygon will decrease the shadow counter (leaving the shadow volume)…
- The shadow counter starts with:
  - 0 if viewpoint is outside the shadow volume
  - 1 if viewpoint is inside the shadow volume





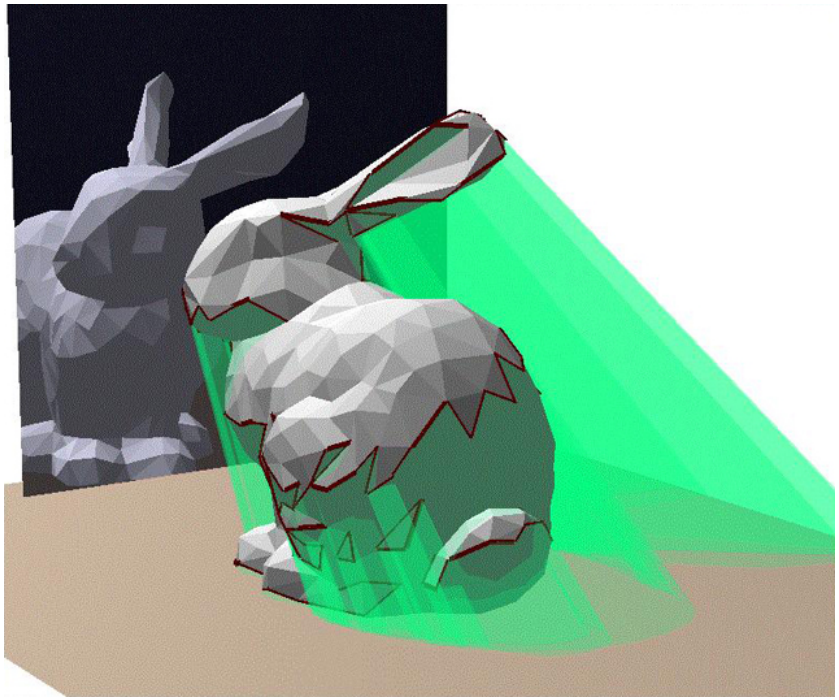
Shadow Volumes computation

- All front facing polygons (from the light point of view) are shadow volume polygons (the front cap)
- To compute the side walls of the shadow volume we need to determine which edges of the B-rep are silhouette edges.
- A silhouette edge is an edge connecting two faces of the model such that:
  - the normal of one of the faces points towards the light source
  - the normal of the other face points away from the light source
- A boundary edge is also a silhouette edge
- If the vertices of a silhouette edge are Va and Vb, we extrude the edge away from the light. Determine two additional points V'a and V'b, outside the view volume such that L (the light position), Va and V'a are co-linear, as well as L, Vb and V'b
- Finally connect al V'* points to form the other cap of the volume



Stenciled Shadow Volumes (91)

1. Render the scene as if it were completely in shadow
2. For each light source construct a mask in the stencil buffer with holes where the visible surface is not inside a shadow volume
3. Render the scene again as if it were completely lit. Stenciled pixels will get this render results added to the shadowed rendering of the first pass.



- There are 3 variations on how to produce the mask in step 2: depth-pass, depth-fail and exclusive-or

Stenciled Shadow Volumes: Depth pass*

1. Disable z-buffer and colour buffer
2. Turn back-face culling on
3. Set stencil operation to increment on depth pass (count shadow volume polygons in front of the object)
4. Render shadow volumes (only front faces will be drawn)
5. Turn front-face culling on
6. Set stencil operation to decrement on depth pass
7. Render shadow volumes (only back faces will be draw)

\+ Shadow volume doesn't need to be capped at the rear end

\- Doesn't work if viewpoint is inside shadow volume (light behind object or object casting shadow behind the viewpoint)

* algorithm starts with an unlit version of the model already rendered, including the depth buffer values

Stenciled Shadow Volumes: Depth fail*

1. Disable z-buffer and colour buffer
2. Front-face culling on
3. Set stencil to increment on depth fail (count shadows behind the object
4. Render shadow volumes
5. Back-face culling on
6. Set stencil operation to decrement on depth fail
7. Render shadow volumes

\+ Works for viewpoints inside shadow volumes too because shadow surfaces between eye and object are not counted

\- shadow volume needs to be capped at the rear

* algorithm starts with an unlit version of the model already rendered, including the depth buffer values

Shadows

Stenciled Shadow Volumes: Exclusive or*

1. Disable z-buffer and colour buffer
2. Set stencil operation to XOR on depth pass
3. Render shadow volumes

\+ Saves a rendering step

\- Does not work for intersecting shadow volumes

* algorithm starts with an unlit version of the model already rendered, including the depth buffer values

Shadows

2 pass shadow generation with z-buffer (image precision)

1. Generate a view from the light and keep the z-buffer result
2. Use a modified z-buffer algorithm to compute the image from the viewer:
    1. For every visible pixel, transform to light coordinate system (x',y',z') and use (x',y') to access the light z-buffer value z_L.
    2. If z_L is nearer the camera than z' than there is some other object blocking the light and the pixel is in shadow. Otherwise it is lit.

Shadow map

Shadows

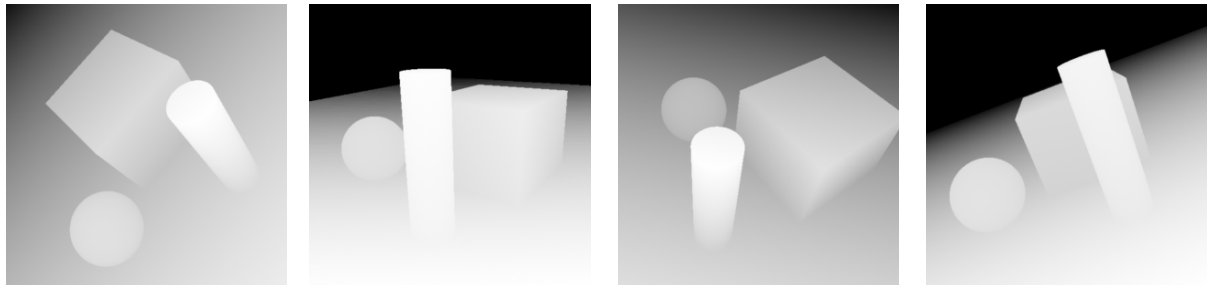
2 pass shadow generation with z-buffer (image precision)

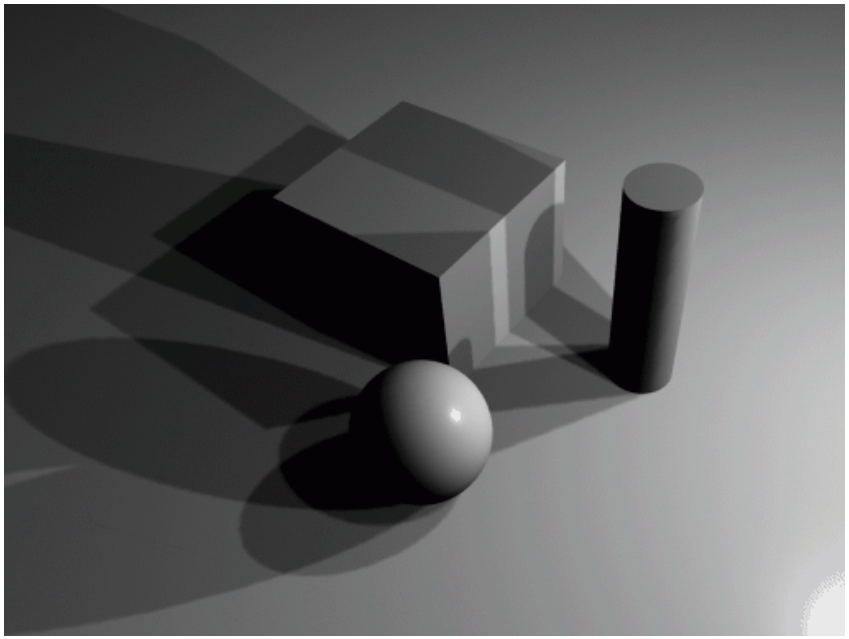
- Several light sources can be handled. For each light source generate a shadow map (z-buffer from light's point of view)

- More adequate for spotlights or lights at infinity

- Aliasing artefacts (shadow acne)

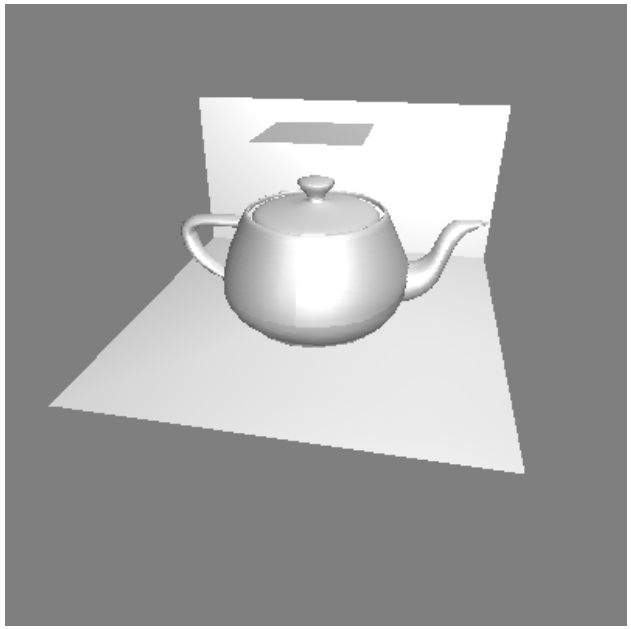
✓ Hardware accelerated



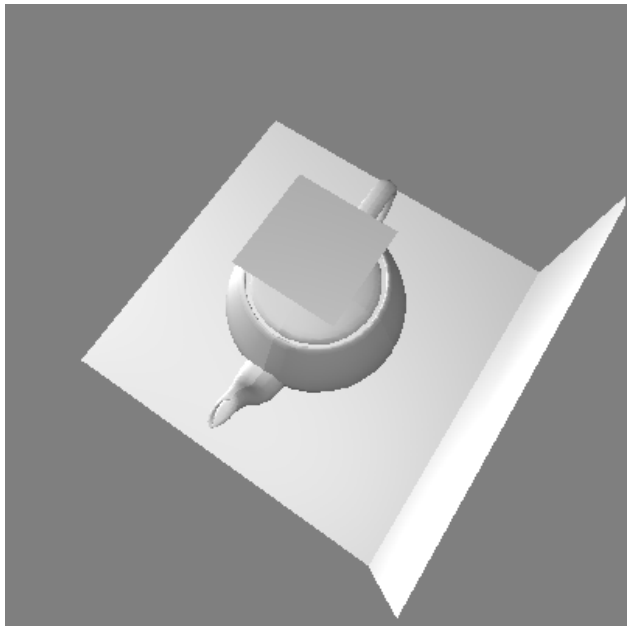


Multiple shadows

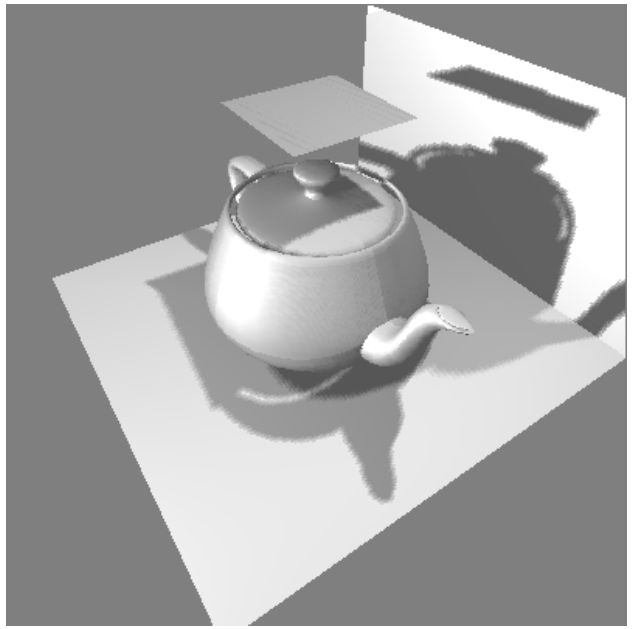
2 pass shadow generation with z-buffer (image precision)



View from light 1



View from light 2

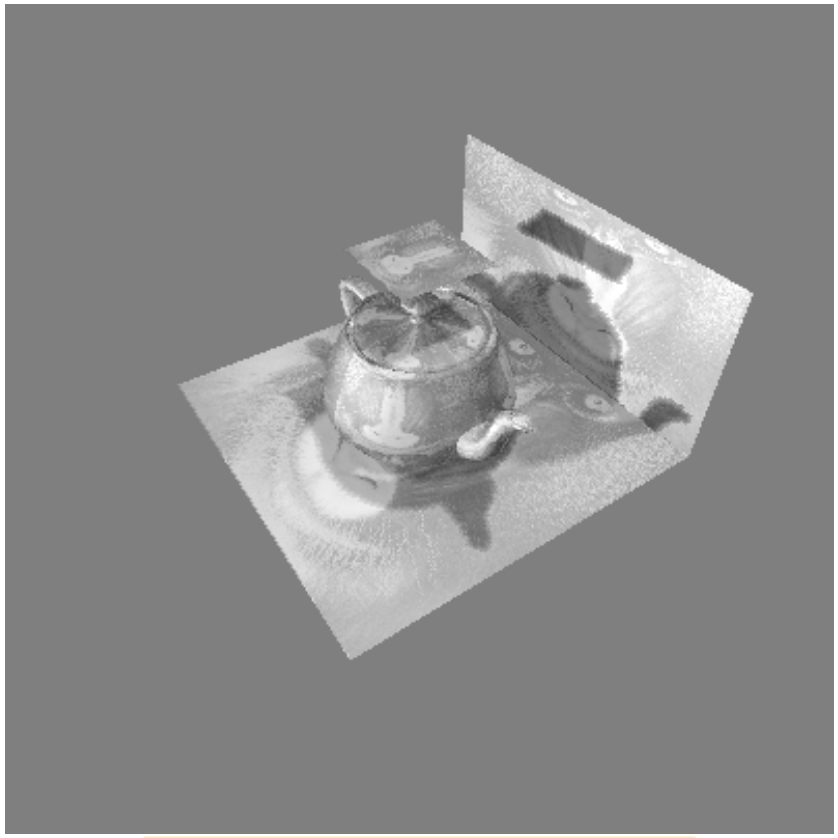


View from viewer

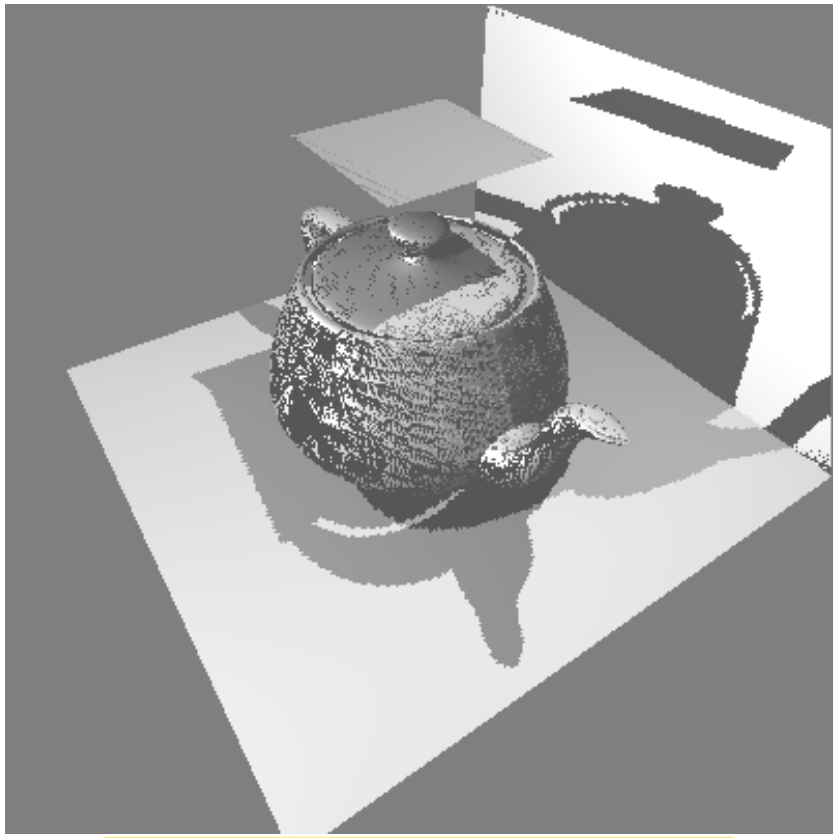
Example from http://web.cs.wpi.edu/~matt/courses/cs563/talks/shadow/shadow.html

Shadows

2 pass shadow generation with z-buffer (image precision)



Shadows on texture mapped objects



Wrong shadows from precision problems

Mapping Applications

Property	Map	Technique
kd, diffuse reflectivity	UV	Diffuse detail mapping, like the upholstery on a sofa
ks, glossy reflectivity	UV	Glossy detail, like the part of a tarnished doorknob that's polished by constant use
Lin	Reflection	Environment mapping
Lout	UV	Light mapping. Texture mapping is used to specify the emissivity of an object like a neon lamp
Position or normal vector	UV	Bump mapping or displacement mapping
Visibility of a light source	Perspective projection	Shadow mapping
Artistic Lout	Various dot products	XToon shading in expressive rendering