IMPORTANT NOTE:

Date, time, and room for the final exam will change!

But I don't know how

→ watch the official website!

Practicals: fill up rooms in that order: 452 - 402 (-406)

Graphics 2008/2009, period 1

Lecture 11

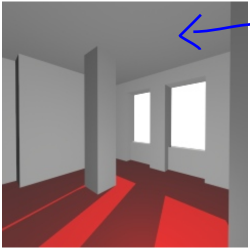
Radiosity and shadows

Note: some of the images used here
have been taken from the following website:
http://freespace.virgin.net/hugo.elias/radiosity/radiosity.htm
(copyright restrictions may apply)

Tutorial

Direct vs. global lighting

Direct lighting:



Global lighting:

Ray tracing

Ray tracing traditionally considers diffuse reflections only in the local lighting calculations.

The recursive step only deals with perfect specular reflection.

Monte-Carlo methods approximate global diffuse reflections, but if the sampling density is low, then the results are noisy, and if the sampling density is high, the method takes lots of time.

Ambient shading

Parts of diffusely reflecting objects that are not directly lit by a light source appear completely black in traditional ray tracing, unless we resort to a standard trick: ambient lighting.

Remember the basic idea: simulate “global light” by adding a constant color term to the color of each object, i.e.

$$c = c_r c_l \max(0, l \cdot n)$$

becomes

$$c = c_r(c_a + c_l \max(0, l \cdot n))$$

Limitations: moving an object close to a bright surface has no effect, and color bleeding does not occur.

View dependent vs. independent lighting calculations

Moreover, ray tracing is a view dependent rendering method. Moving the virtual camera even just slightly means that we have to recompute the entire image.

Diffuse reflections are view independent. If only there was a way to compute diffuse global illumination for diffuse reflections . . .

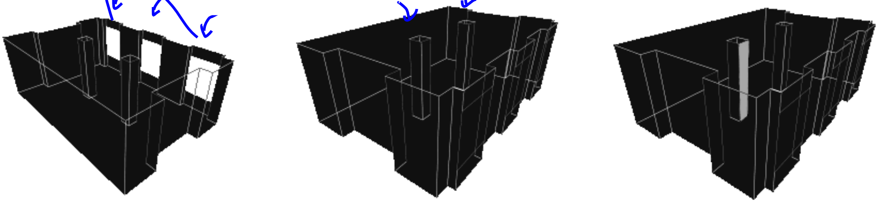
For scenes with mainly diffusely reflecting objects, we could pre-compute the diffuse global illumination, and do walk-throughs in real time.

We could also replace the ambient lighting in ray tracing by more realistic global diffuse illumination.

Radiosity

Such a method for global illumination with diffuse reflections only exists: Radiosity.

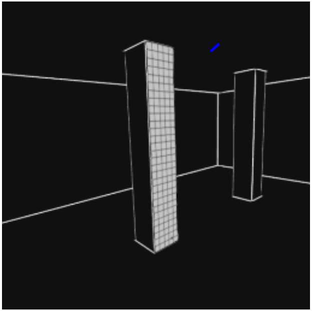
Example: light

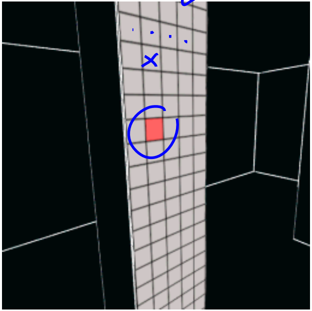


Radiosity

thermal heat transfer

Radiosity doesn't compute light transport along a discrete number of rays, but instead focuses on energy transfer between patches, into which the polygons in the model are subdivided.





Radiosity equations

We define the radiosity of a patch as the amount of energy that leaves the patch per unit time per unit area. It is measured in W/m^2.

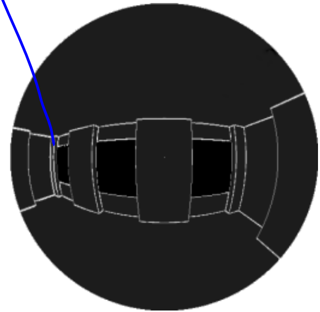
Informally, the radiosity of a patch is a measure for its brightness.

The radiosity of a patch not only depends on the energy it emits, light but also on the energy that it reflects (where does this energy come from?) reflections from incoming light

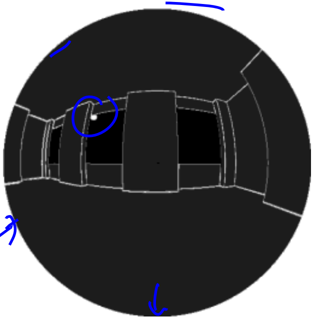
In the radiosity method, every patch can be a light source; there are no separate point light sources.

Radiosity

View from a patch

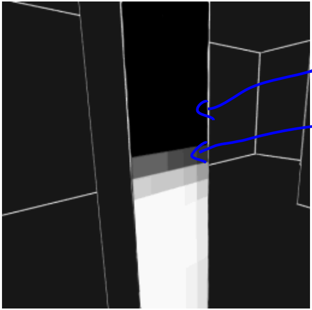


View from a lower patch

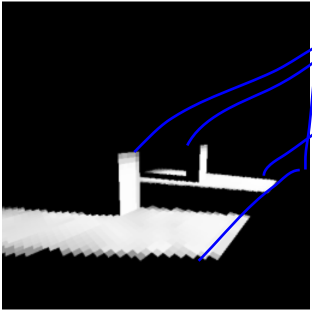


Radiosity

Pillar and ...



... entire room ...



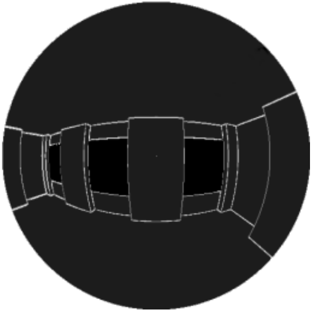
... after emission from the direct light source is considered.

But what about the reflections from enlighted objects?

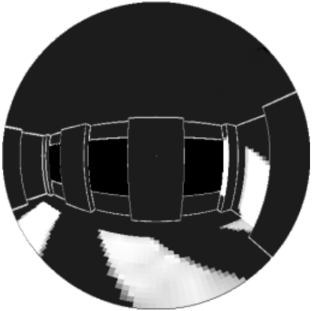
Radiosity

Compare view from the upper patch ...

... before ...



... and after ...

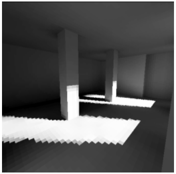


... emission from the direct light source is considered.

Radiosity

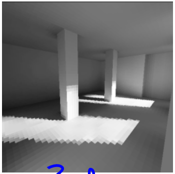
Ah, there's more light! Let's consider that as well ...

... and again



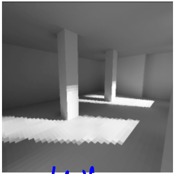
2nd

... and again



3rd

... and again

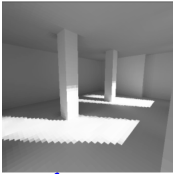


4th

...

...

... and again



16th pass

... until it converges (or we are satisfied with the result).

Radiosity equations

Now, if we denote the radiosity of patch A_i with B_i, the energy it emits by E_i, and its reflectivity by ρ_i, then we can write

Radiosity B_i of patch A_i

Emitted energy

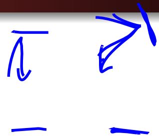
Reflected energy

$$B_i = E_i + \rho_i \sum_j B_j F_{ji} \frac{A_j}{A_i}$$

$(j \neq i)$

where F_{ji} depends on the shapes of patches A_i and A_j, their distance, their orientation, etc: not all energy that leaves A_j reaches A_i. F_{ji} is called the form factor from A_j to A_i.

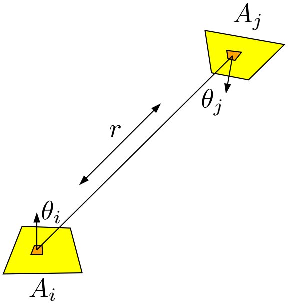
Form factors



- orientation
- size
- distance

Form factors are dimensionless, and specify the fraction of the energy leaving one patch that arrives at the other patch. They are given by

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \, \cos\theta_j}{\pi r^2} \, dA_j \, dA_i$$



There's symmetry in form factors:

$$A_i F_{ij} = A_j F_{ji}$$

$F_{ji} = F_{ij} \cdot \frac{A_i}{A_j}$

Q: What if A_i and A_j are not (completely) mutually visible?

$F_{ij} = 0$

Solving the radiosity equations

Recall that the radiosities of the patches are given by

$$B_i = E_i + \rho_i \sum_j B_j F_{ji} \frac{A_j}{A_i}$$

Because of the symmetry relation we can rewrite this to

$$B_i = E_i + \rho_i \sum_j B_j F_{ij}$$

And so

$$B_i - \rho_i \sum_j B_j F_{ij} = E_i$$

(Note that we have n of equations in n variables.)

Solving the radiosity equations

$$B_i - \rho_i \sum_j B_j F_{ij} = E_i$$

If we know the emissions E_i of all patches, and all form factors F_{ij}, we can solve for the B_i's:

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_1 F_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

Solving the radiosity equations

So, we just have to:

- Compute all the form factors

  $F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \, \cos\theta_j}{\pi r^2} dA_j \, dA_i.$

- Solve the (very large) system of linear equations on the previous slide

Hmm, "just"?

Computing form factors analytically

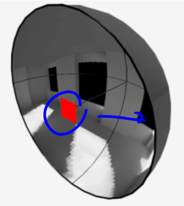
For any pair of patches A_i and A_j we have to compute the form factors

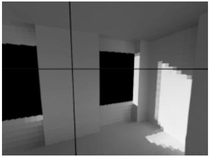
$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \, \cos\theta_j}{\pi r^2} dA_j \, dA_i$$

In theory this can be done analytically, but in practice this is too complicated, especially if we have to take care of partial occlusion of patches.

The hemisphere method

Nusselt showed that computing F_{dij} for a differential patch dA_i is equivalent to projecting A_j onto a unit hemisphere centered about dA_i, projecting the projected area orthographically onto the hemisphere's unit base circle, and dividing by the area of the circle.

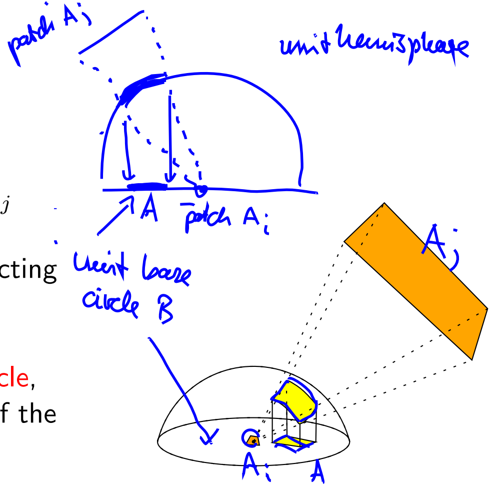




The hemisphere method

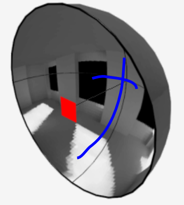
Nusselt showed that computing F_{dij} for a differential patch dA_i is equivalent to projecting A_j onto a unit hemisphere centered about dA_i, projecting the projected area orthographically onto the hemisphere's unit base circle, and dividing by the area of the circle.

form factor $F_{dij} = \frac{A}{B}$

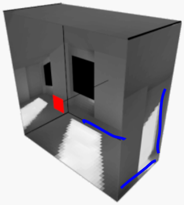


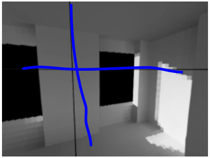
The hemisphere method

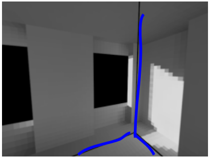
Hmm, but that still seems like a lot of work.



$\approx$

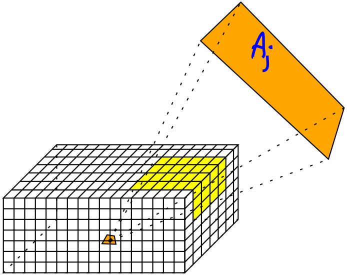






The hemicube method

Instead of projecting analytically onto a hemisphere, we can also project onto a hemicube that is subdivided into cells, each cell having a weight that represents its contribution to the form factor. By summing the weights of the cells onto which A_j is projected, we approximate the form factor. This can be implemented on standard graphics hardware. ⇝ fast! :)



Number of form factors

Note that the number of form factors is quadratic in the number of patches. This means that in practice, especially with large models, storing all form factors is impossible.

Therefore, form factors are (re)computed on the fly when they are needed.

Progressive refinement

Now that we know how to compute form factors, we are done halfway. What is left is the computation of the radiosities. Theoretically, we could solve the system

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_1 F_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

In practice, this is too expensive, and we have to resort to approximation methods.

Progressive refinement

For each patch, we have to compute $B_i = E_i + \rho_i \sum_i B_j F_{ji}$. We approximate this iteratively:

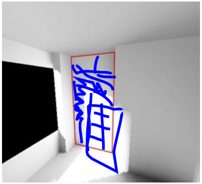
- Initially, set $B_i = E_i$ for every patch.
- For every patch A_i compute the energy reaching it from all other patches and add this (multiplied by ρ_i) to the radiosity of patch A_i that has been computed so far.
- Repeat the previous step, but only account for the unshot radiosity that was added to each patch A_j in the previous iteration.
- Repeat until the added radiosity per iteration is less than a threshold for each patch.

Meshing

We subdivide polygons into patches because in general there is a smooth variation of the radiosity along a polgon. The finer the subdivision, the more accurate the results.

On the other hand, a uniform subdivision into very small patches leads to very long computation times.

A fine subdivision is not necessary everywhere: if the distance between patches A_i and A_j is large, the variation of the contribution of A_i to the radiosity of A_j over the surface of A_j will be small.



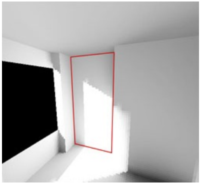
Adaptive subdivision

Adative subdivision is an iterative process:

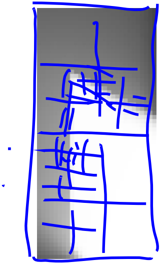
- Make an initial (coarse) subdivision of polygons into patches, and compute the radiosities.
- Check neighboring patches. If their radiosities differ more than a threshold, subdivide the patches into sub-patches (also called elements).
- Compute the radiosities of the elements by computing the influence of other patches (as opposed to elements).
- Repeat until the radiosities of neighboring patches differs less than a threshold, or element sizes have reached a pre-determined minimum.

Adaptive subdivision

Example for adative subdivision:



$\longrightarrow$



From radiosities to image

Once the radiosities have been computed, we have to generate an image. This can be done by computing radiosities for the vertices of the mesh (e.g., as a weighted average of the surrounding elements), and then applying Gouraud shading.

If the scene is static, we can do walk-throughs in real-time.

Radiosity can also be used as a more realistic replacement of ambient light and local diffuse reflections in ray tracing.

2-pass ~~solution~~ solution

Graphics 2008/2009, period 1

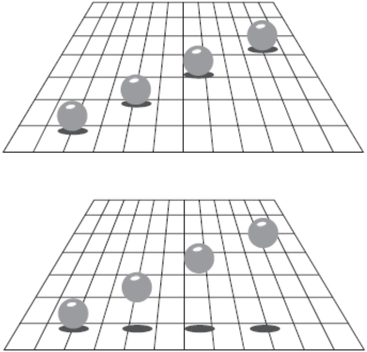
Lecture 11

Radiosity and shadows

The need for shadows

Shadows come “for free” in ray tracing and radiosity methods, but in projective methods they require some extra effort.

However, these efforts are well-spent: shadows add realism and depth, as well as give a better understanding of the images.



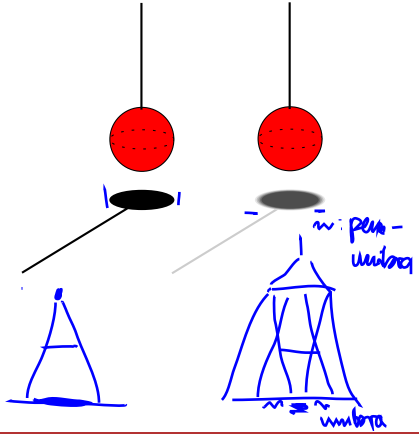
a.21

Hard and soft shadows

Point light sources give hard shadows, whereas area lights give soft shadows.

For soft shadows, the region that receives no light at all is called the umbra, and the region that receives partial light is called the penumbra.

The penumbra gives additional cues with respect to the size of the light source and the distance from the occluder to the receiver.



Shadow maps

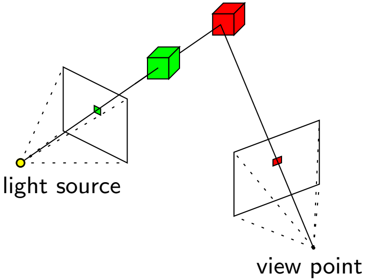
1st approach

fast!

A popular hardware supported shadow rendering algorithm uses shadow maps.

The scene is first rendered from the light source point-of-view to fill a depth buffer, stored with the light.

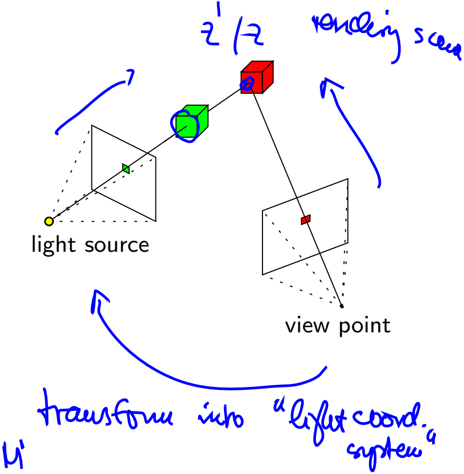
z-Buffer



Shadow maps

Next, the scene is rendered from the camera point-of-view. The z-coordinate rendered polyon for a pixel is converted with a transformation matrix to a z-coordinate in the light source coordinate system.

If the light source z-buffer has a smaller z-value stored, then the part of the polygon projected to the current pixel lies in shadow.



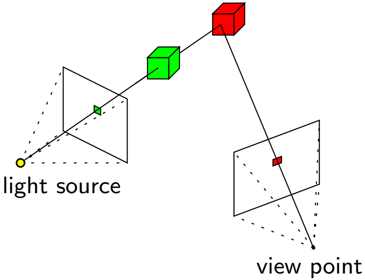
Shadow maps

Advantages:

- hardware-supported (z-buffer and matrix ops), so very fast.

Disadvantages:

- Suited for spot lights, not really for omni-directional lights.
- Precision problems due to use of floating points and discrete sampling.

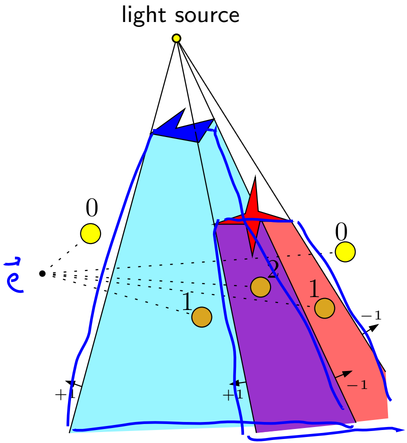


Shadow volumes

2nd appr.

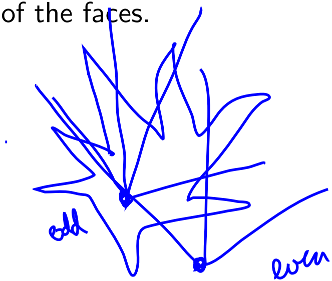
An alternative method for shadow computation uses shadow volumes.

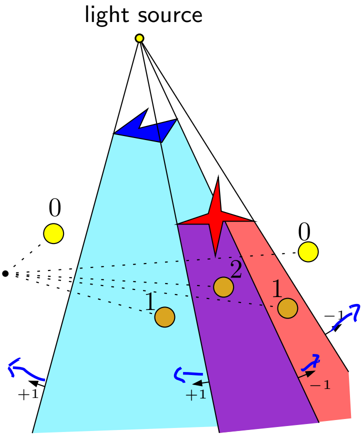
For a light source and an occluder, the 3D shadow volume is determined by the silhouette edges of the occluder w.r.t the light source.



Shadow volumes

The faces of the shadow volumes are either front facing or back facing w.r.t the camera. This is easily determined with the normals of the faces.

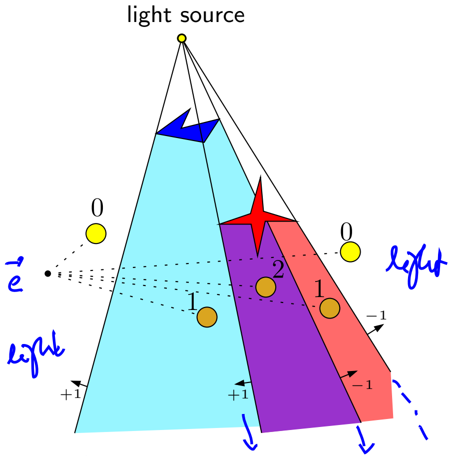




Shadow volumes

The idea: when a pixel is rendered, we follow the ray from the viewpoint to the object for the pixel.

Whenever the ray crosses a shadow volume face, we increment or decrement a counter, depending on whether the face is front-facing or back-facing.

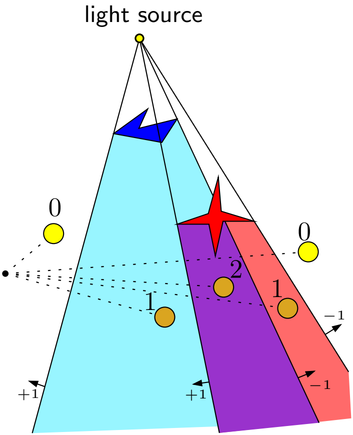


Counter: $\underline{0} \to +1 \to +2 \to +1 \to 0 \to$

>0

Shadow volumes

Objects for which the counter is zero are lit; objects for which the counter is positive are in shadow.



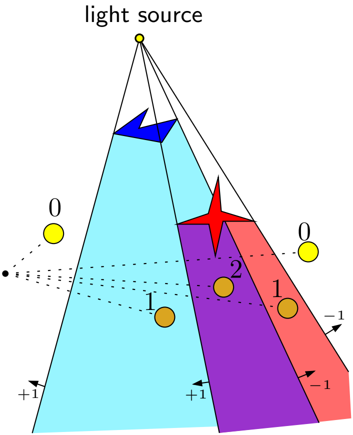
Hardware support: stencil buffer

Obviously, we don't want to do ray tracing to determine shadows in real-time rendering methods.

slow :(

This is where the stencil buffer comes in.

HW :)



Hardware support: stencil buffer

The stencil buffer is comparable to the z-buffer (i.e., there is a one-to-one correspondence of pixels in the frame buffer and entries in the stencil buffer), but here every entry is a counter.

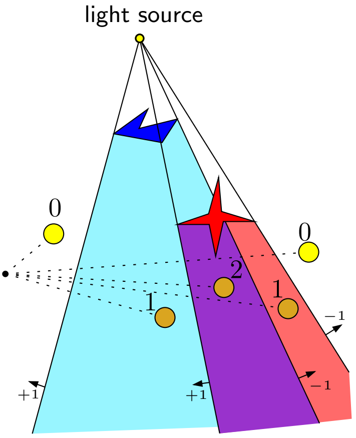
It supports =0 +1 −1

- resetting, incrementing and decrementing the counters.
- idem, but conditionally, depending on a test against the z-buffer.
- conditional drawing in the frame buffer, i.e., only draw a pixel if the corresponding counter is zero/non-zero.

Hardware support: stencil buffer

Computing shadows using the stencil buffer:

- Draw the scene with ambient lighting only (i.e., everything is drawn as if it is in shadow).
- Compute the shadow volumes
- "draw" the shadow volume faces in the stencil buffer.

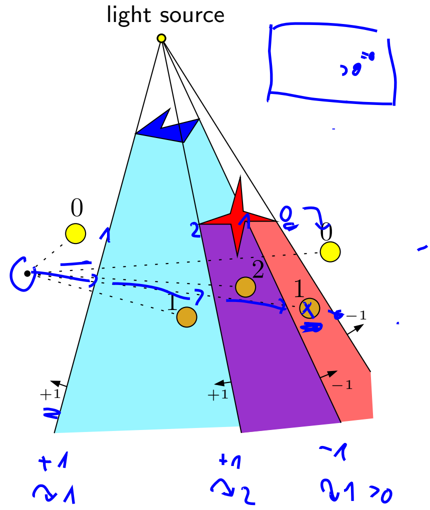


Hardware support: stencil buffer

Drawing a pixel in the stencil buffer means: Test the shadow face "pixel" agains the z-buffer.

IF its z value is smaller than that of the drawn (real) object, THEN

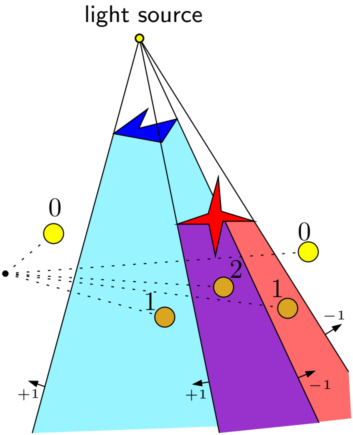
- IF it is a front-facing face, THEN increment the counter.
- IF it is a back-facing face, THEN decrement the counter.



Hardware support: stencil buffer

Once all shadow volume face have been drawn, the stencil buffer contains zeros for pixels that are not in shadow.

Next, draw the entire scene again (only the real objects), this time including lighting calculations, but tell the graphics card to ignore pixels that have a non-zero stencil buffer entry.

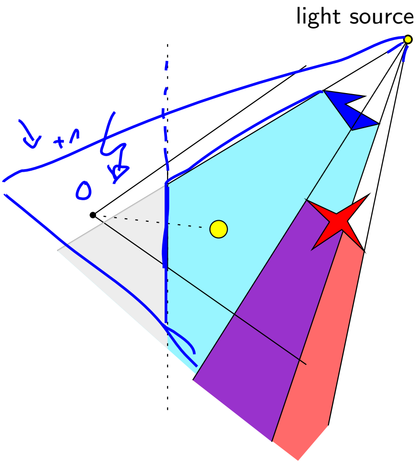


conditional drawing

Shadow volumes: problems

Problems with the shadow volume approach:

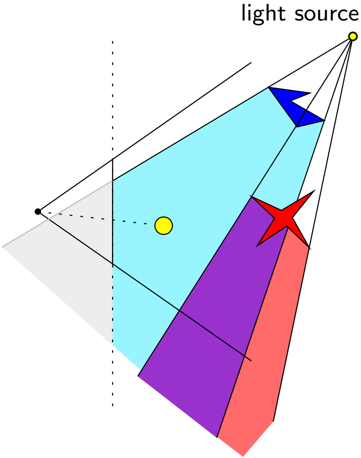
- The view point may lie in one or more shadow volumes.
- The near clipping plane may slice open shadow volumes.



Shadow volumes: problems

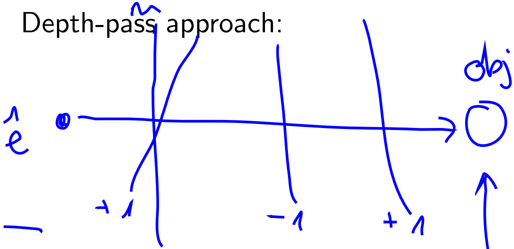
The described problems can be solved by capping the shadow volumes: the near clipping plane (partially) becomes one of the faces of each volume. Drawback: precision problems close to near clipping plane.

The described algorithm uses a depth-pass approach: only shadow volume faces in front of objects are drawn in the stencil buffer.



Shadow volumes: depth-pass vs. depth-fail

Depth-pass approach:



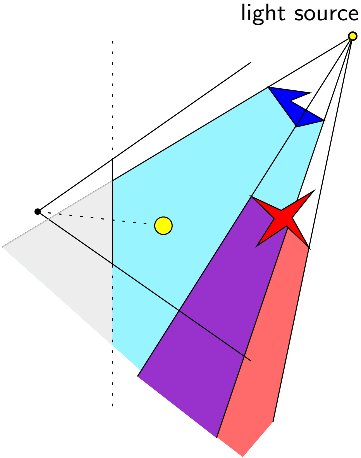
0 = light

>0 = shadow

Depth-fail approach:

Shadow volumes: problems

Alternative: depth-fail approach: only draw/count shadow volume faces behind object. Problems with near clipping plane are traded for problems with far clipping plane.

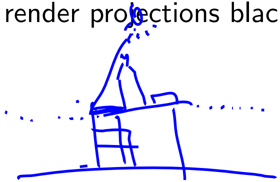


Fake shadows

3rd app.

A fast and easy (but not extremely realistic) way of creating shadows: fake shadows. Only few (important) occluders are selected, and shadows are projected onto supporting planes of only few (important) surfaces (e.g.: table has shadows, but chairs around it haven't).

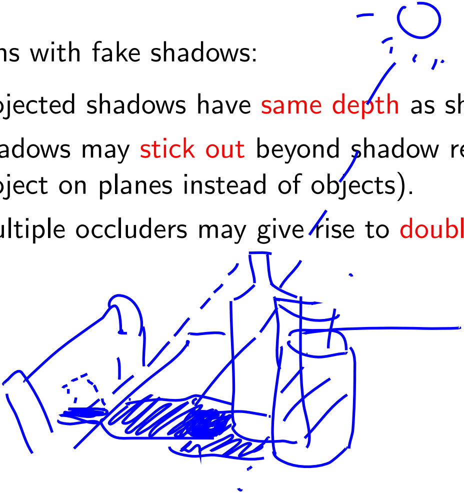
- Draw the scene, including lighting.
- Project occluders onto designated planes. Improvement: don't render projections black, but blend with original colors.



Fake shadows

Problems with fake shadows:

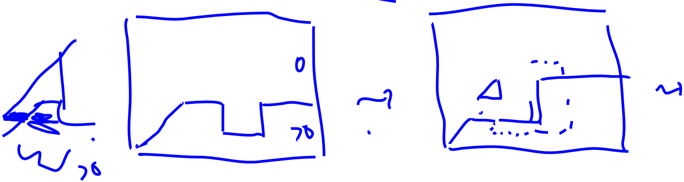
- projected shadows have same depth as shadow receivers.
- Shadows may stick out beyond shadow receivers (since we project on planes instead of objects).
- Multiple occluders may give rise to double blending.



Fake shadows

Solution to the problems with fake shadows: use stencil buffer.

- Reset stencil buffer counters. = 0
- Draw scene. When shadow receivers are drawn, increment corresponding stencil buffer counters.
- Project shadow polygons, but only draw/blend for pixel that have a non-zero stencil value. Reset stencil entry after drawing, to avoid double blending.



Soft Shadows

Soft shadows are expensive. Usual approach: approximate by sampling area lights, treating every sample as a point light source. Compute individual shadows, and average.

Drawback: many samples are needed to get smooth penumbras, and this hurts performance.