Image Processing and Computer Graphics

Transparency and Reflection

Matthias Teschner

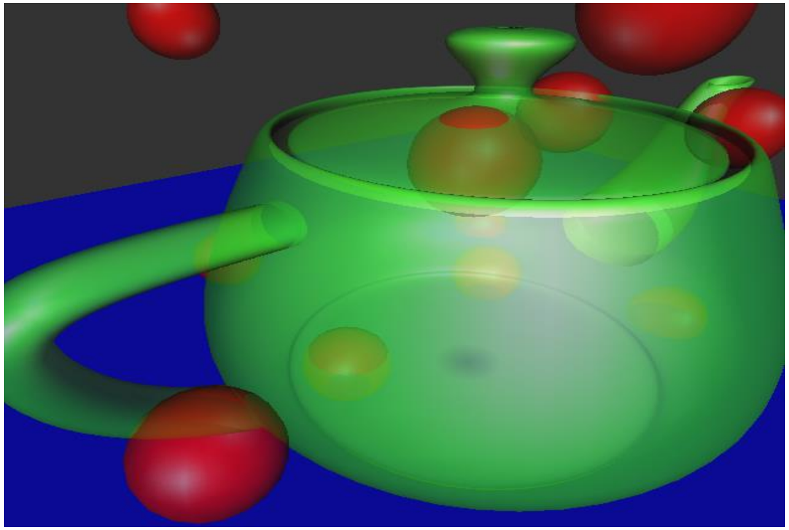
Computer Science Department
University of Freiburg

Outline

- transparency
- reflection

Introduction

- simplified transparency model
  - semitransparent objects are filters / attenuators of occluded objects
  - refraction and object thickness are neglected
- algorithms are based on
  - stipple patterns
  - color blending per pixel



[Cass Everitt: Interactive Order-Independent Transparency]

Stipple Patterns

- screen-door transparency
- transparent object is rendered with a fill / stipple pattern, e.g. checkerboard (pattern of opaque and transparent fragments)
- limited number of fill patterns results in limited number of transparency levels
- aliasing artifacts
- simple method

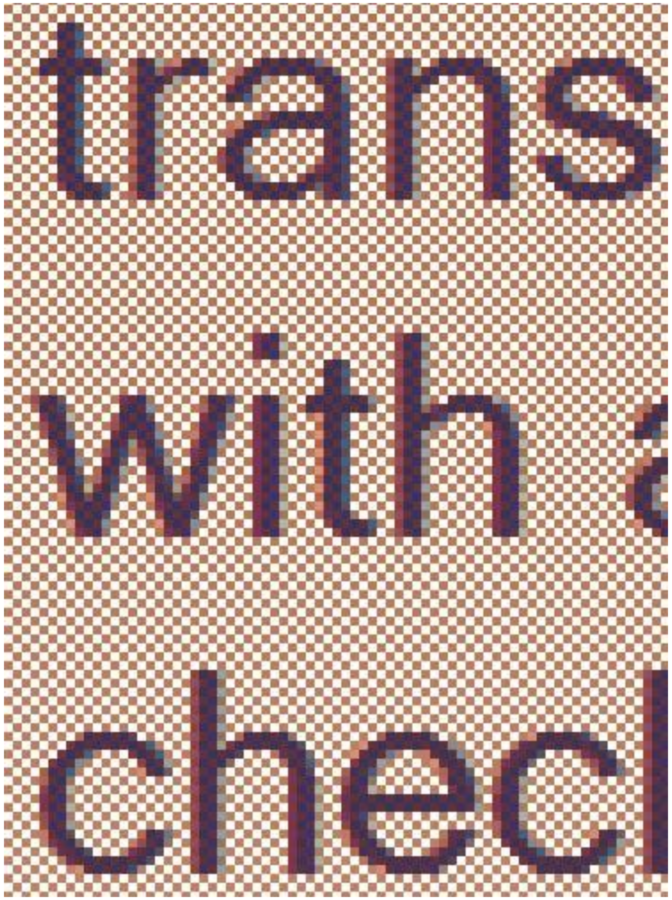
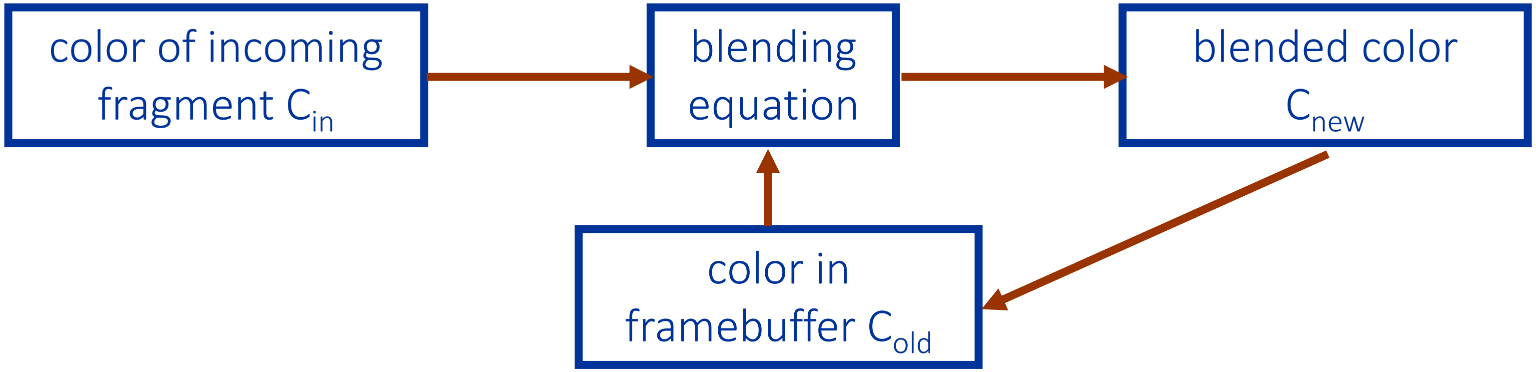


illustration of a stipple pattern

Color Blending

- combine fragment color with the framebuffer content



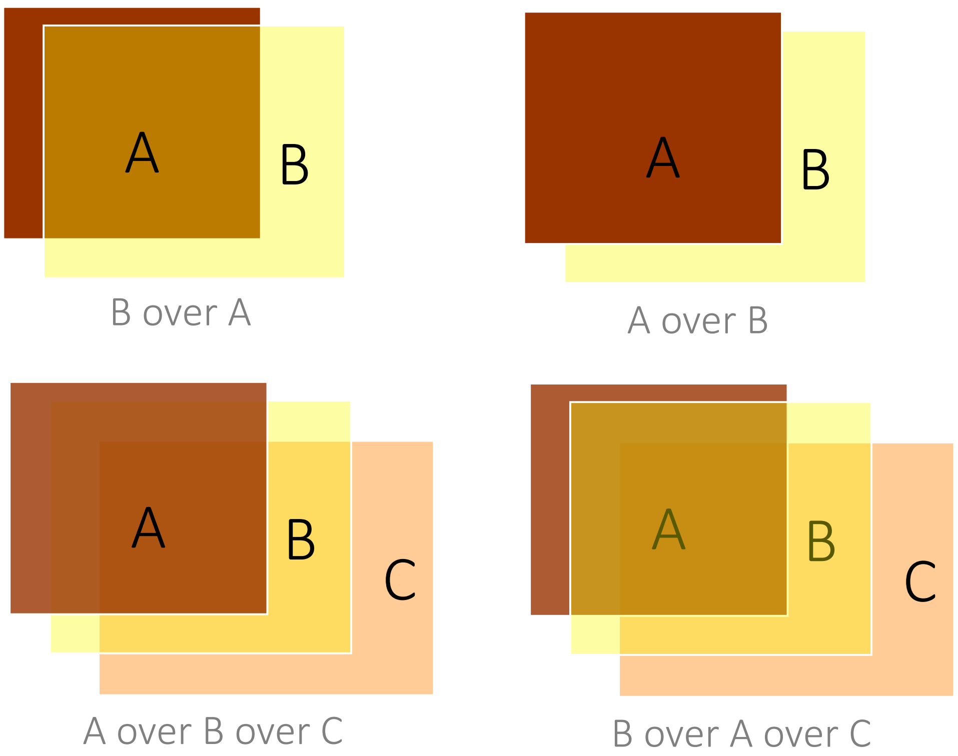
- color C_{old} is replaced by C_{new}
- blending equation: $\mathbf{C}_{\text{new}} = \alpha_{\text{in}} \cdot \mathbf{C}_{\text{in}} + \alpha_{\text{old}} \cdot \mathbf{C}_{\text{old}}$

Color Blending

- alpha value
  - describes the opacity of a fragment, 1 - opaque, 0 - transparent
  - stored together with RGB color in a 4D vector (RGBA)
- blending equation for transparency
  - $\mathbf{C}_{\text{new}} = \alpha_{\text{in}} \cdot \mathbf{C}_{\text{in}} + (1 - \alpha_{\text{in}}) \cdot \mathbf{C}_{\text{old}}$
  - over operator
  - $\alpha_{in} = 0$ - C_{old} is not changed
  - $\alpha_{in} = 1$ - C_{old} is replaced by C_{in}
  - $0 < \alpha_{in} < 1$ - C_{old} is replaced by a mix of C_{in} and C_{old}
  - only the alpha value of the incoming fragment matters

Color Blending

- order matters

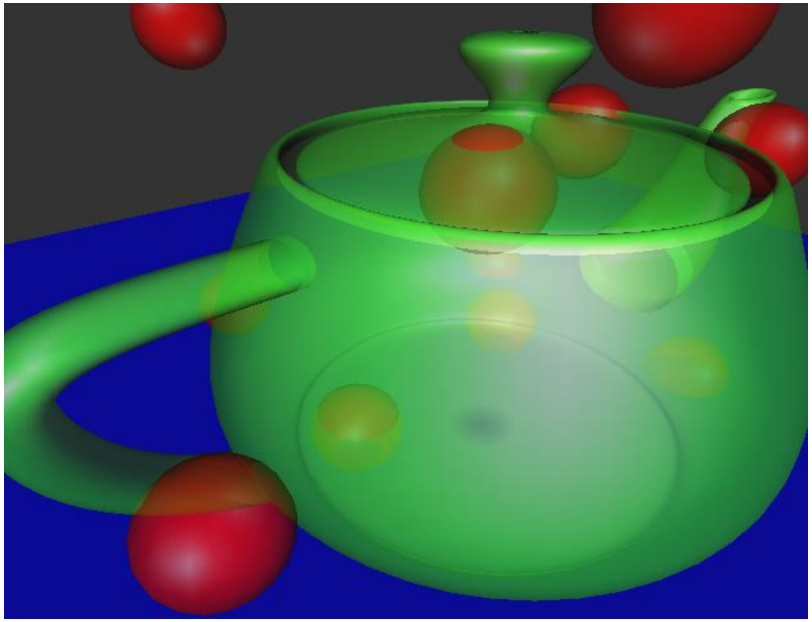


Outline

- transparency
  - depth ordering
  - binary space partitioning
  - depth peeling
- reflection

Depth Ordering

- polygons / fragments have to be rendered in sorted depth order
- hardware generally renders in object order
- depth test only returns the nearest fragment per pixel, sorting is not realized
- intersecting polygons have to be handled
- dynamic scenes require re-sorting



[Cass Everitt: Interactive Order-Independent Transparency]

Depth Ordering for Convex Objects

- exactly two depth layers for arbitrary viewing directions
- first depth layer defined by front faces
- second depth layer defined by back faces
- algorithm
  - render back faces in a first pass
  - blend with front faces in a second pass

Depth Ordering for Arbitrarily Shaped Objects

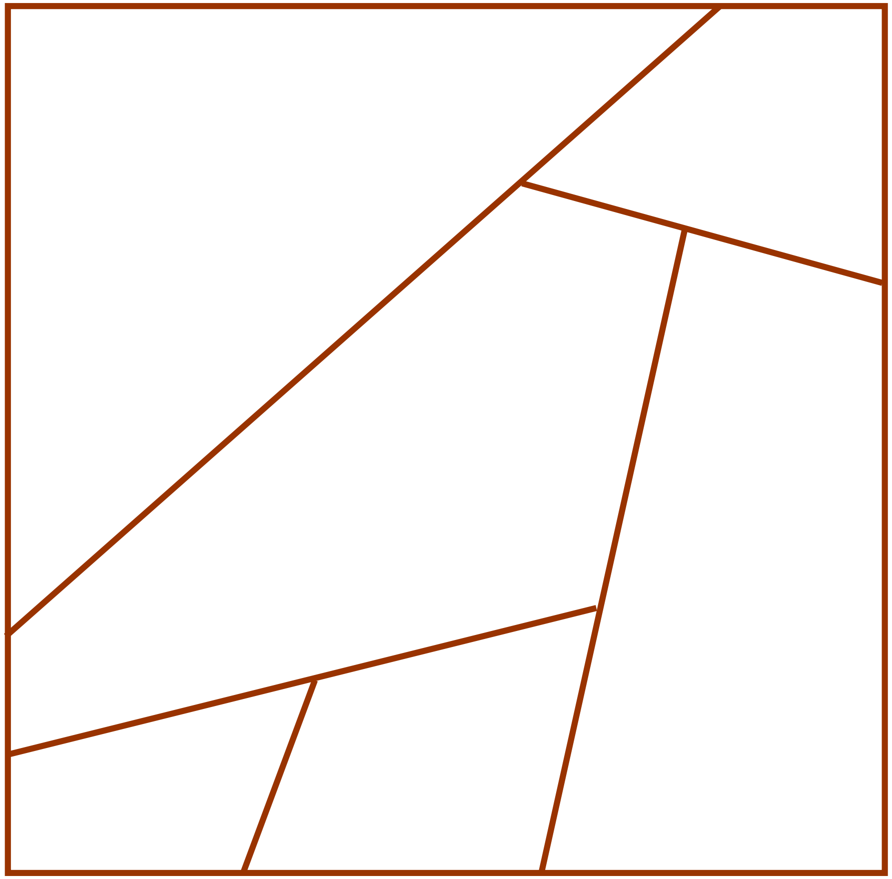
- object-space methods
  - use pre-computed spatial data structures
  - e.g., binary space partition tree (BSP tree)
  - useful for static geometry
  - varying viewer positions and orientations can be handled
- screen-space methods
  - employ the functionality of the rendering pipeline
  - several rendering passes compute depth layers
  - final pass renders the ordered depth layers
  - useful for dynamic / deforming geometry and arbitrary views
  - no pre-computation is required / can be employed

Outline

- transparency
  - depth ordering
  - binary space partitioning
  - depth peeling
- reflection

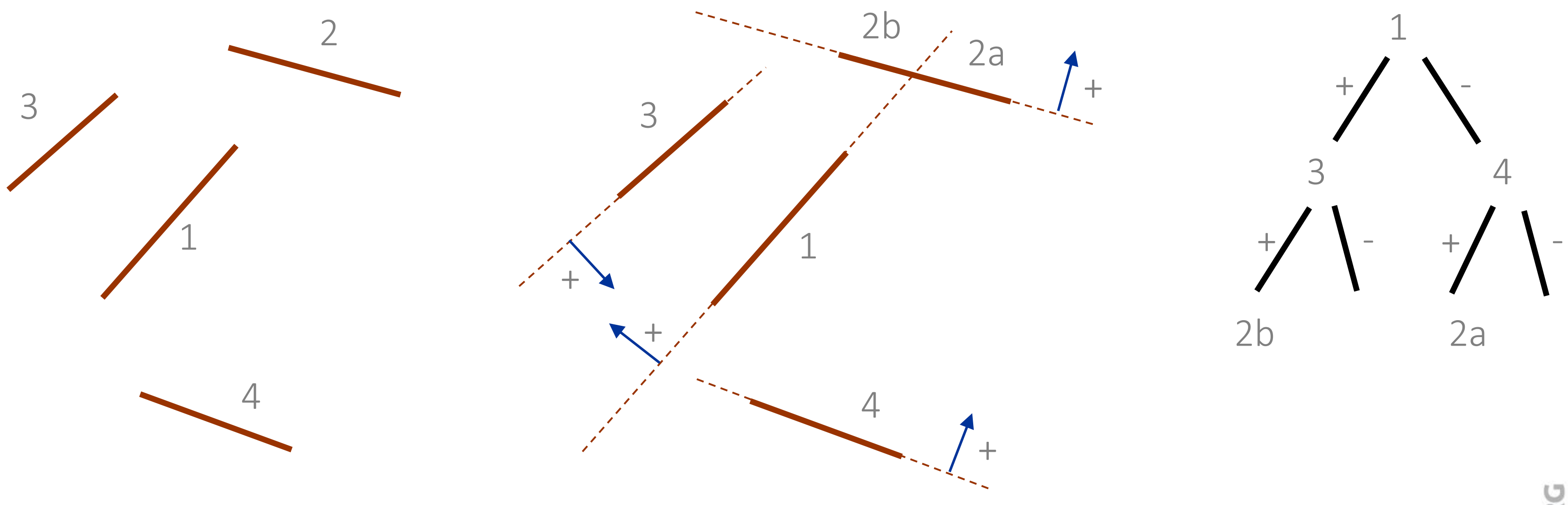
Binary Space Partitioning BSP

- BSP tree is a hierarchical spatial data structure
- 3D space is subdivided by means of arbitrarily oriented planes
- nodes represent planes
- leaves represent convex space cells
- applications
  - visible surface algorithm
  - depth sorting
  - collision detection



Generation of the BSP Tree

- the BSP tree is pre-computed for static scenes
- all planar primitives are represented in the tree
- balancing is less important, as the entire tree has to be queried (all primitives are rendered)



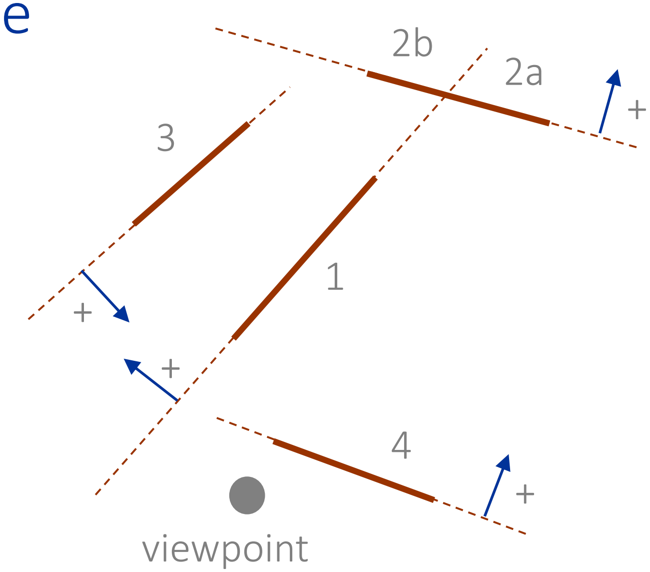
2D scene

space partitioning

BSP tree

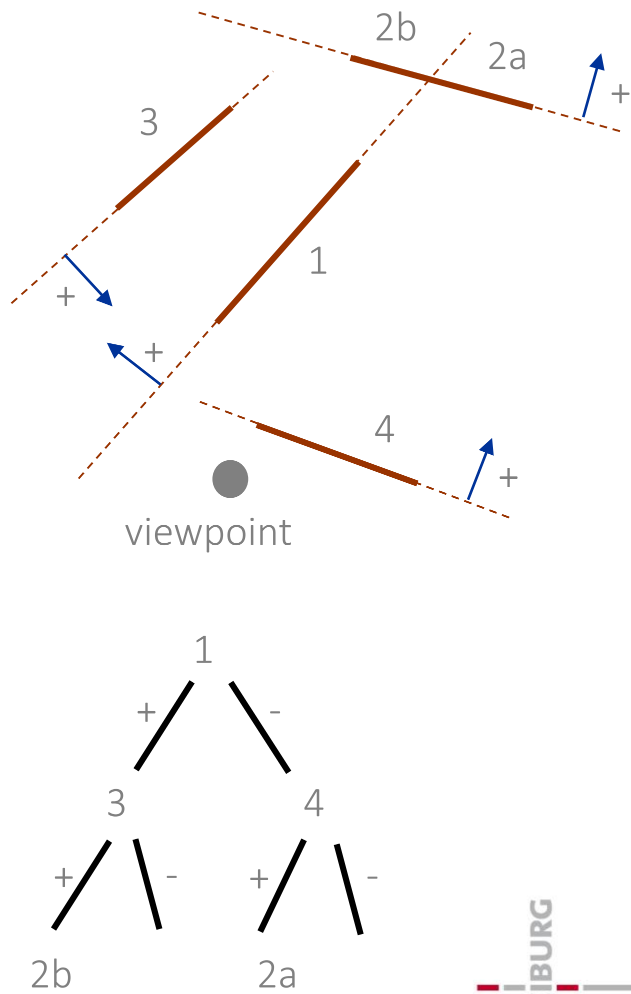
Query of the BSP Tree

- motivation
  - a viewer is on the near side of a plane
  - a polygon on the far side of this plane cannot occlude the plane or any polygon on the near side
- back to front rendering
  - render far branch of the viewpoint
  - render root (node) polygon
  - render near branch of the viewpoint
  - recursively applied to sub-trees



Query of the BSP Tree

- back to front rendering
- viewpoint is in 1-
- rendering of 1+, 1, 1-
- rule recursively applied to 1+ and 1-
- viewpoint is in 3+
  - rendering of 3, 2b
- viewpoint is in 4-
  - rendering of 2a, 4



BSP Tree - Discussion

- not only visible surface generation, but depth sorting of all primitives per pixel position
- additional data structure
- can be pre-computed
- requires polygon splits
- dynamic scenes require an update of the data structure

Outline

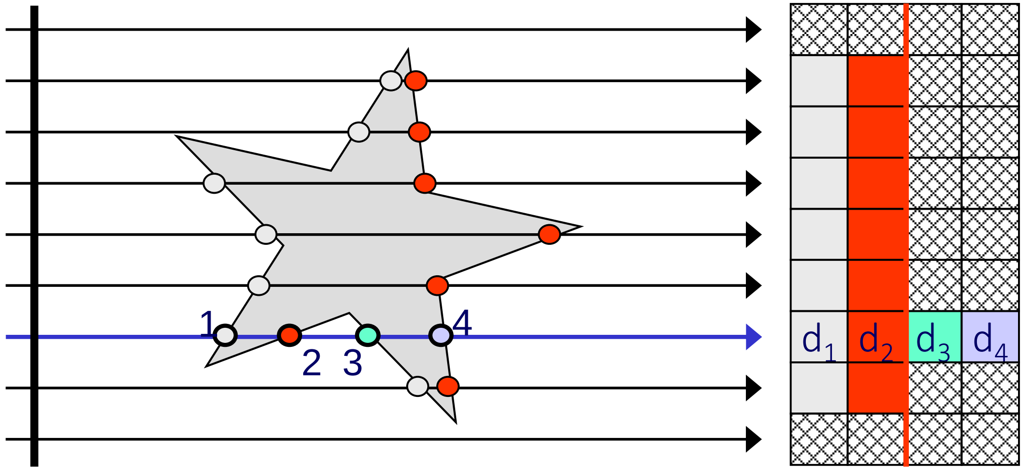
- transparency
  - depth ordering
  - binary space partitioning
  - depth peeling
- reflection

Concept

- motivation
  - use the functionality of the rendering pipeline
  - several rendering passes compute depth layers
  - final pass renders the ordered depth layers
  - useful for dynamic / deforming geometry
  - no pre-computation is required / can be employed
- algorithm
  - first render pass gives the front-most fragment color / depth
  - each successive render pass extracts the fragment (with color and depth) for the next-nearest fragment on a per pixel basis (screen-space approach)
  - two depth buffers are used

Concept

- object is rendered once for each depth layer
  - depth complexity is the max number of layers per pixel position
- two separate depth tests per fragment
  - must be farther than the one in the previous layer (d_1)
  - must be the nearest of all remaining fragments (d_2, d_3, d_4)

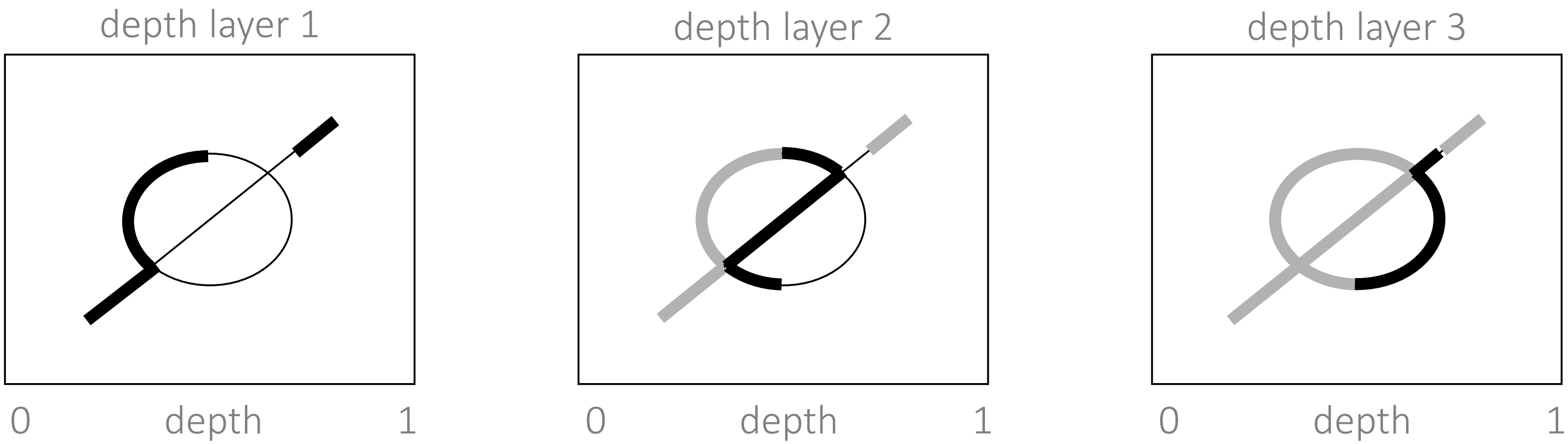


[Bruno Heidelberger]

Depth Layers - 2D

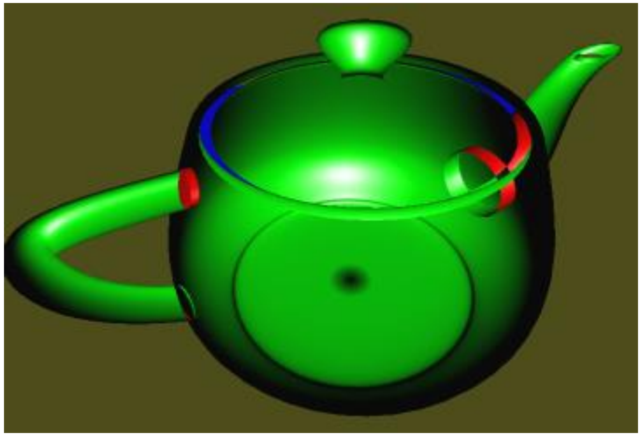
- depth peeling strips away one depth layer with each successive rendering pass
- illustration
  - bold black lines - frontmost (leftmost) surfaces
  - thin black lines – hidden surfaces
  - light grey lines – “peeled away” surfaces

[Cass Everitt: Interactive Order-Independent Transparency]

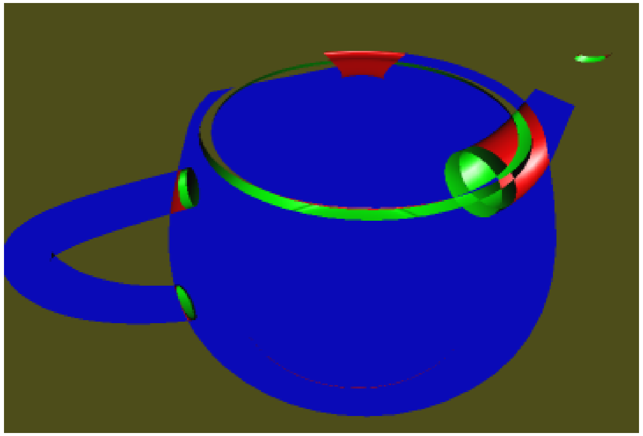


Depth Layers - 3D

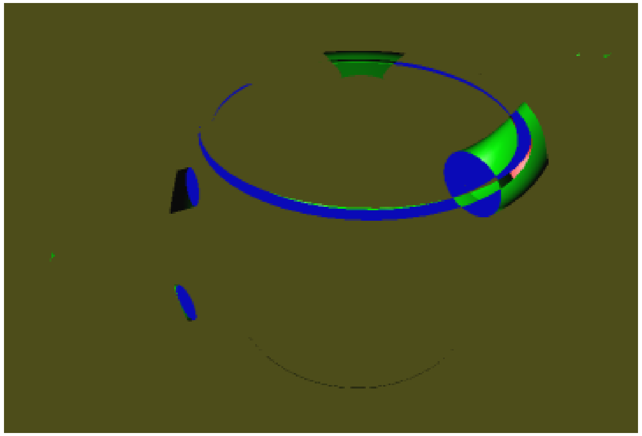
depth layer 1



depth layer 2



depth layer 3



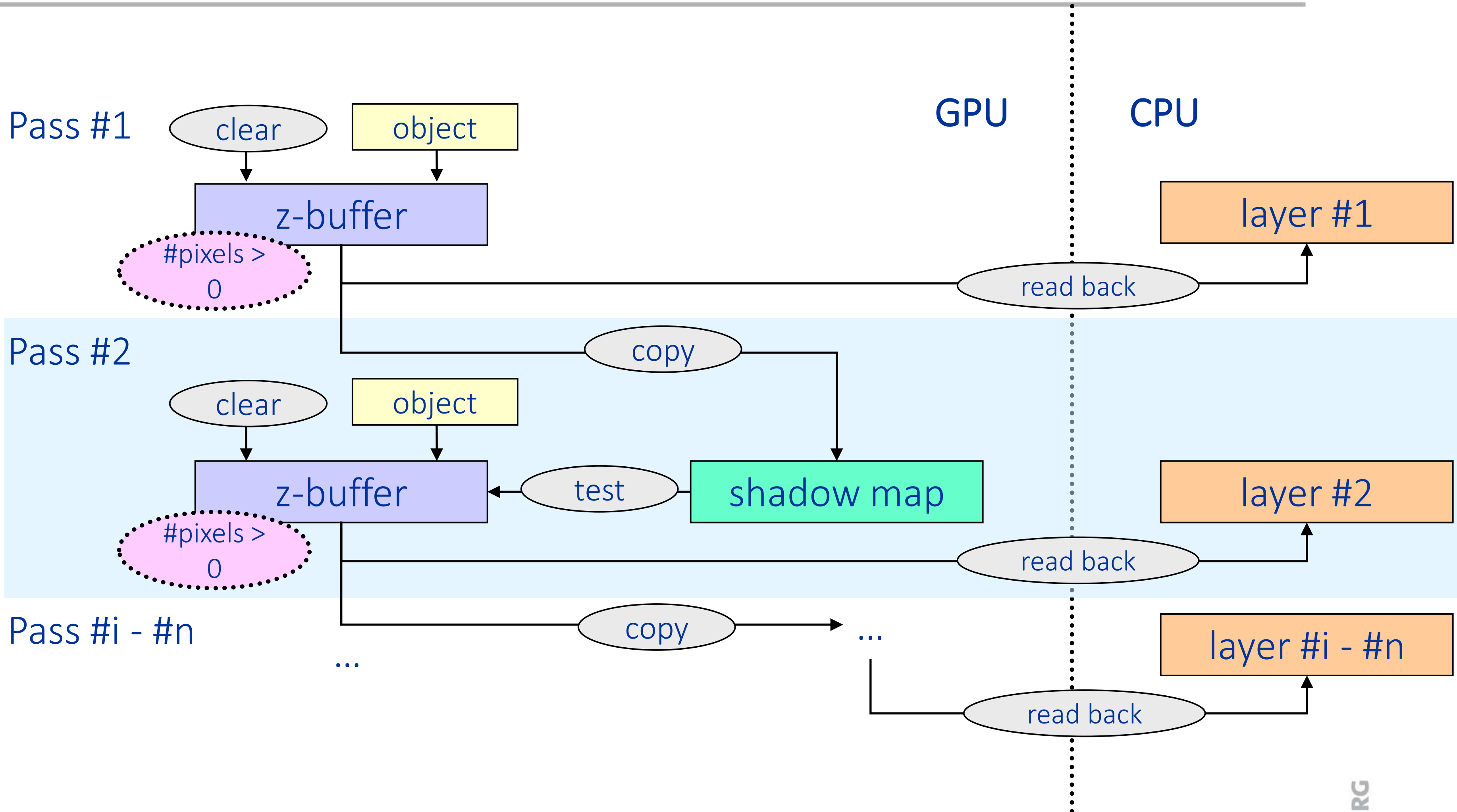
depth layer 4

[Cass Everitt: Interactive Order-Independent Transparency]

Implementation Based on Shadow Mapping

- two depth tests (depth buffers) are required
- e.g., shadow mapping can be used to realize a second depth buffer
- in contrast to the depth buffer, the shadow map
  - is not tied to the camera position
  - is not writeable during depth test
  - does not discard fragments
- with respect to depth peeling
  - the shadow map is tied to the camera position
  - copy functionality to depth buffer is employed

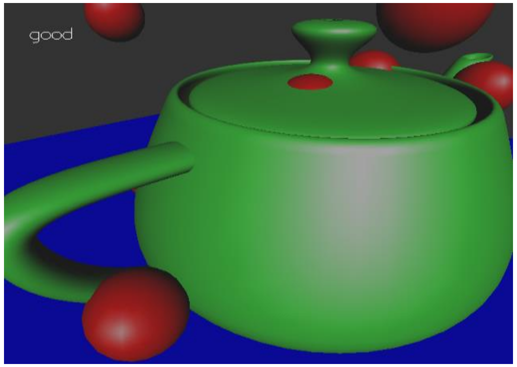
Implementation Based on Shadow Mapping



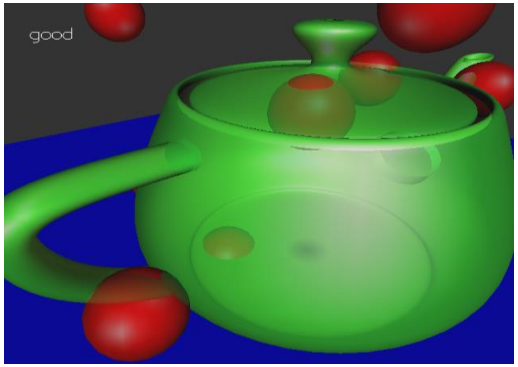
[Bruno Heidelberger]

Results

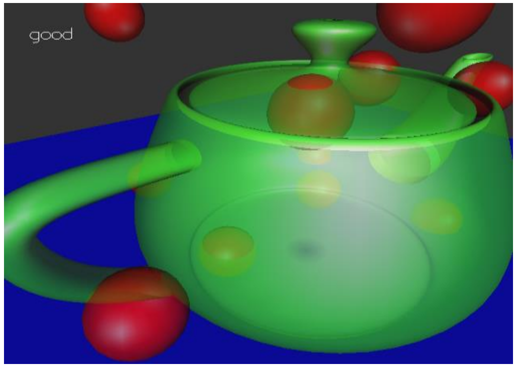
- quality and performance are determined by the number of generated depth layers



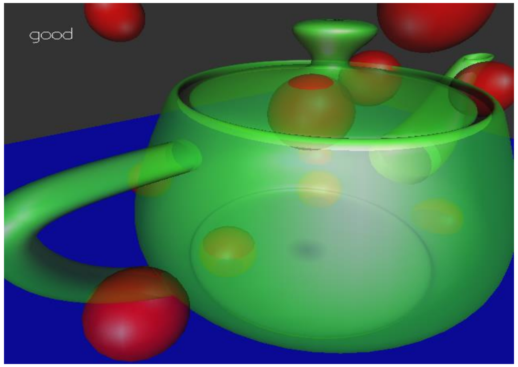
one depth layer



two depth layers



three depth layers



four depth layers

[Cass Everitt: Interactive Order-Independent Transparency]

Depth Peeling - Discussion

- screen-space algorithm
- multiple rendering passes generate depth layers per pixel position
- view dependent (in contrast to the BSP approach)
- appropriate for dynamic scenes
- quality and performance are determined by the number of rendering passes (in the discussed implementation)

Transparency - Summary

- simplified transparency model
- algorithms based on
  - stipple patterns
  - color blending
- for blending, depth-sorted primitives are required
- BSP tree
  - object space algorithm with one rendering pass
  - appropriate for static scenes
- depth peeling
  - screen space algorithm with multiple rendering passes
  - appropriate for dynamic scenes

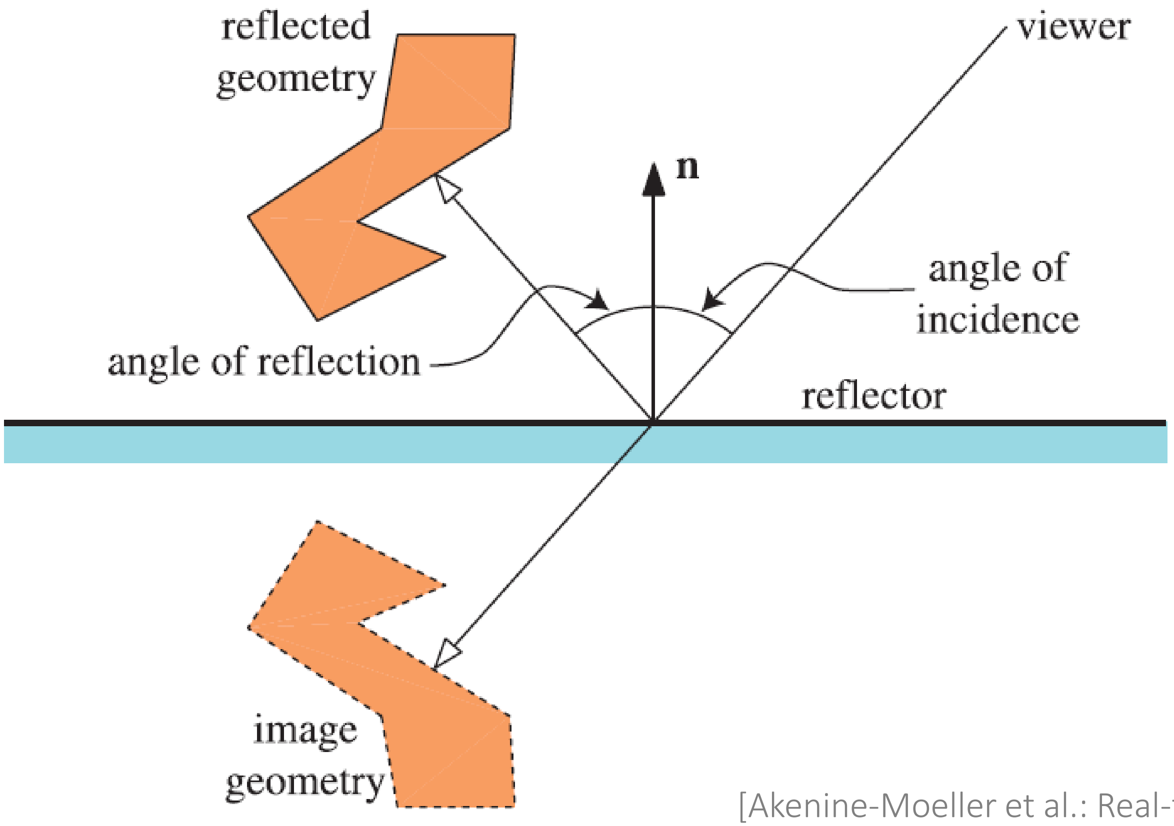
UNI FREIBURG

Outline

- transparency
- reflection
  - planar surfaces
  - arbitrary surfaces

Law of Reflection

- angle of incidence is equal to the angle of reflection



[Akenine-Moeller et al.: Real-time Rendering]

Generation of Reflected Geometry

- original and reflected geometry is rendered
- reflected geometry is generated with respect to the reflection plane with surface normal $\mathbf{n} = (n_x, n_y, n_z)$ and a point $\mathbf{p}$ on the reflection plane

$$\mathbf{M}_{(\mathbf{n},\mathbf{p})} = \begin{pmatrix} 1-2n_x^2 & -2n_x n_y & -2n_x n_z & 2n_x(\mathbf{n}\cdot\mathbf{p}) \\ -2n_x n_y & 1-2n_y^2 & -2n_y n_z & 2n_y(\mathbf{n}\cdot\mathbf{p}) \\ -2n_x n_z & -2n_y n_z & 1-2n_z^2 & 2n_z(\mathbf{n}\cdot\mathbf{p}) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

e.g.

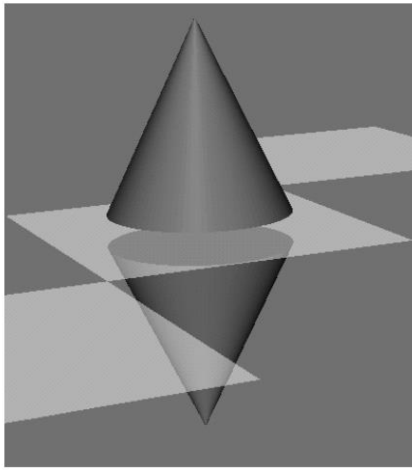
$$\mathbf{M}_{((0,1,0),(0,0,0))} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

UNI FREIBURG

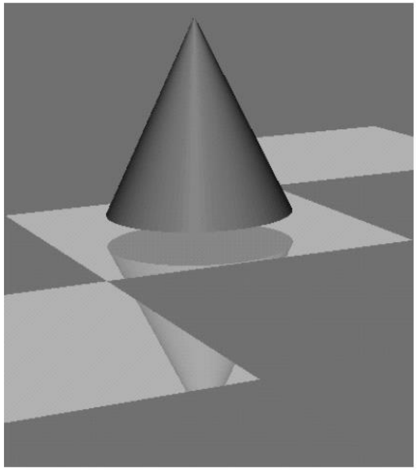
Implementation

- render reflected geometry with reflected illumination
- render semi-transparent reflection plane, e.g. with color blending
- render original geometry

- render reflection plane to stencil
- render reflected geometry where stencil is set
- ...



reflection rendering without stenciling



reflection rendering with stenciling

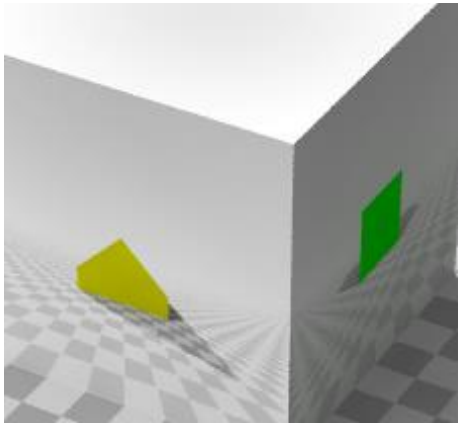
[Akenine-Moeller et al.: Real-time Rendering]

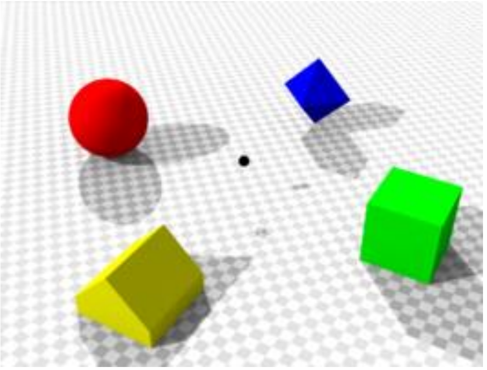
Outline

- transparency
- reflection
  - planar surfaces
  - arbitrary surfaces

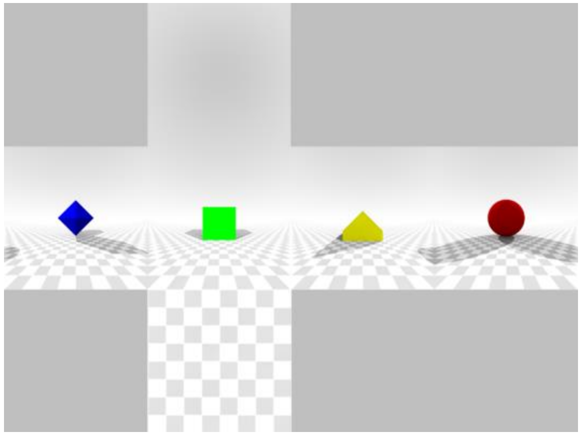
Environment Mapping

- e.g. cube mapping
- approximates reflections of the environment on arbitrary surfaces

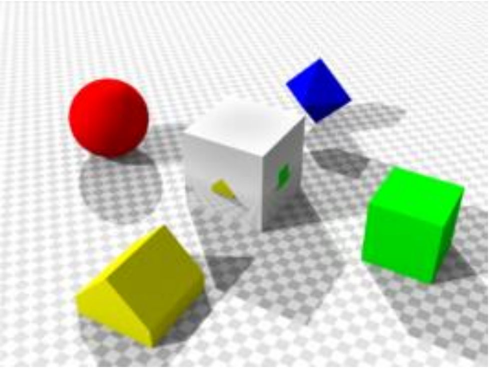




place a viewer in a scene



generate the environment texture from six view directions

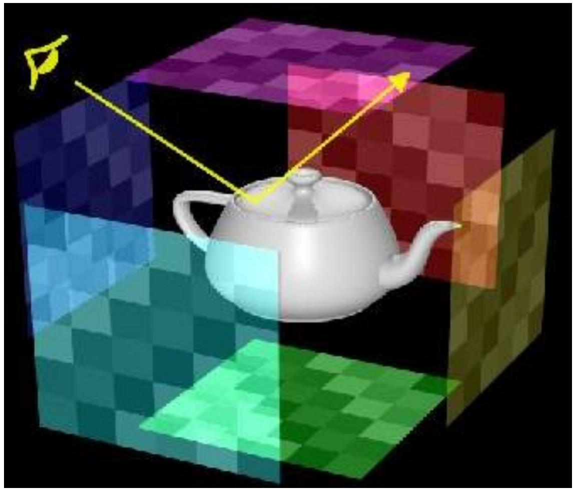


apply the texture to an object at the position of the viewer

[Wikipedia: Cube Mapping]

Environment Mapping

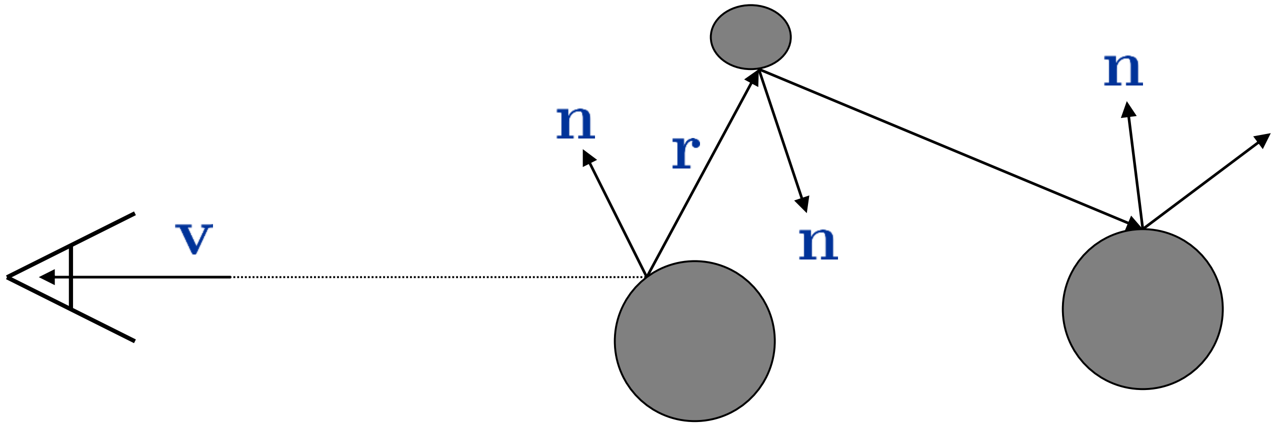
- environment is projected onto an object-embedding shape, e. g. sphere or cube
- view-dependent mapping
  - dependent on viewing and reflection direction
- approximate implementation of reflections off arbitrary surfaces



[Rosalee Wolfe]

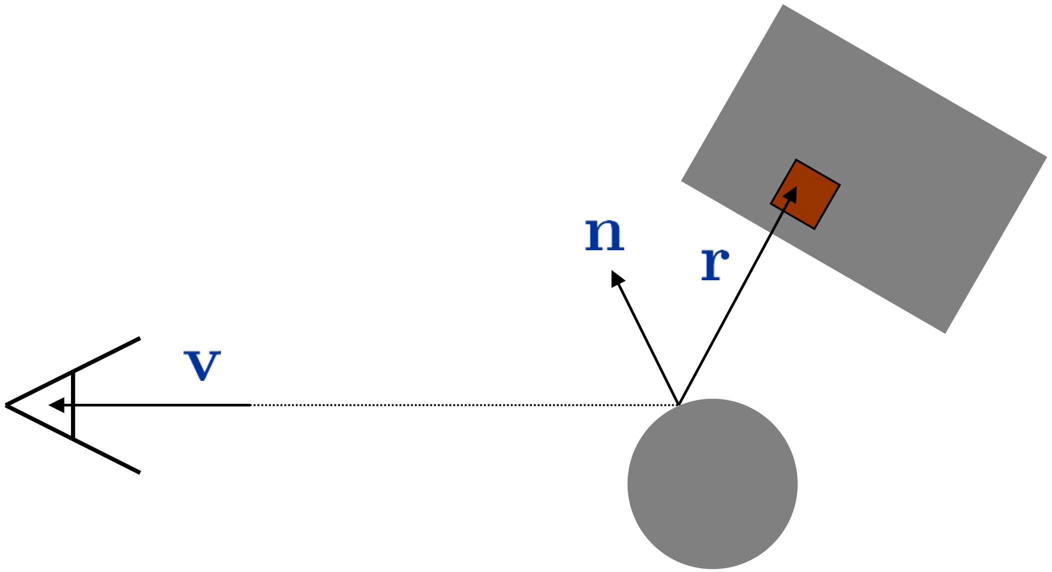
Environment Mapping
Motivation

- the Phong illumination model (a local model) does not take into account reflections
- Raytracing (a global model) traces rays off the object into the world to obtain reflections



Environment Mapping

- environment mapping approximates this process by capturing the environment in a texture map and using the reflection vector to index into this map
- cannot handle changing reflections of moving objects

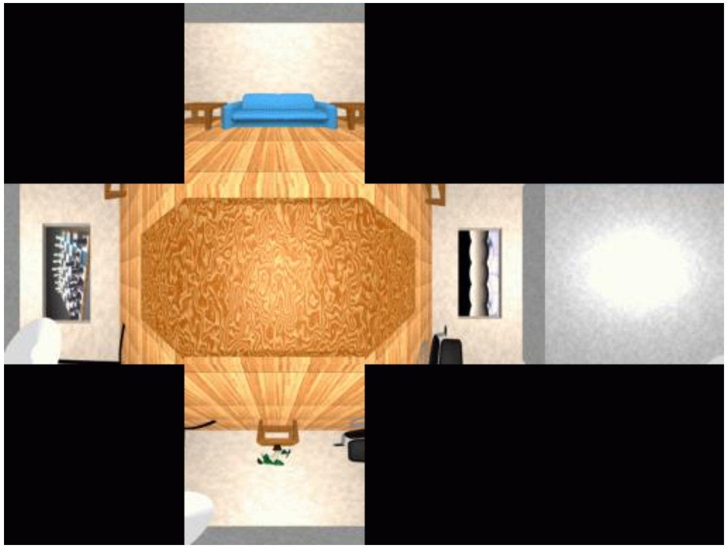


Environment Mapping - Steps

- generate or load a 2D map of the environment
- for each fragment of a reflective object, compute the normal **n**
- compute the reflection vector **r** from the view vector **v** and the normal **n** at the surface point
- use the reflection vector to compute an index into the environment map that represents the objects in the reflection direction
- use the texel data (texture value) from the environment map to color the current fragment

Cubic Environment Mapping

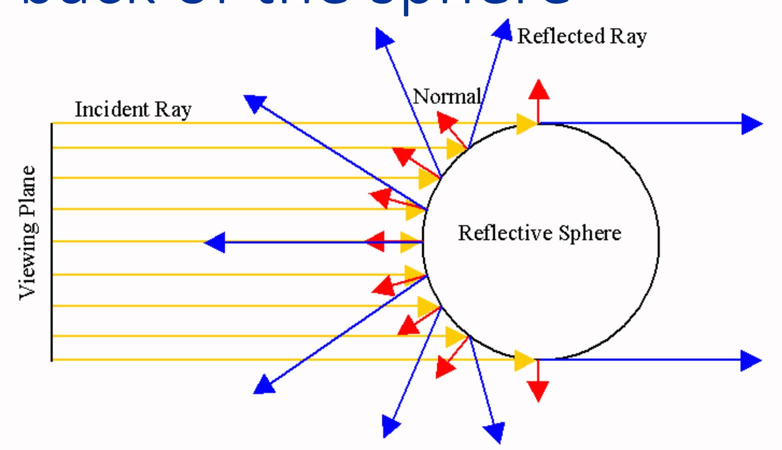
- the map is constructed by placing a camera at the center of the object and taking pictures in 6 directions

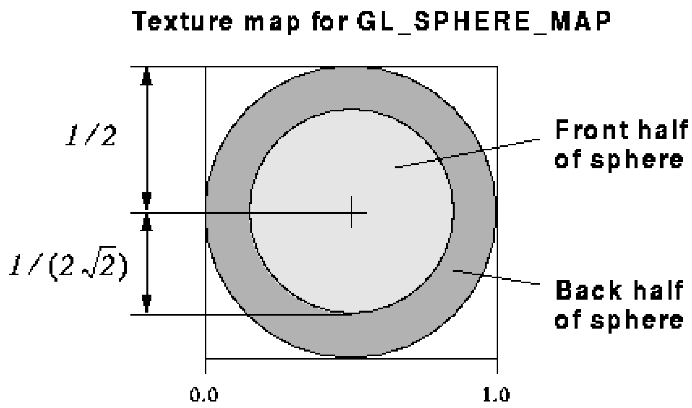


[Mizutani, Reindel: Environment Mapping Algorithms, Reindel Software]

Spherical Environment Mapping

- the map is obtained by orthographically projecting an image of a mirrored sphere
- map stores colors seen by reflected rays
- sphere map contains information about both, the environment in front of the sphere and in back of the sphere

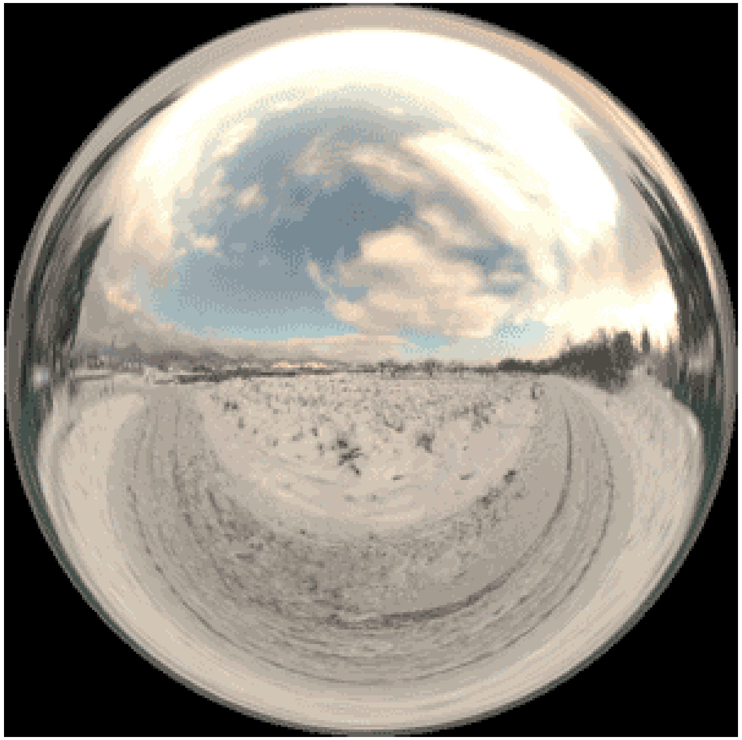




[Mizutani, Reindel: Environment Mapping Algorithms, Reindel Software]

Spherical Environment Mapping

- the map can be obtained from a synthetic scene by
  - Raytracing
  - warping automatically generated cubic maps
- the map can be obtained from the real world by
  - photographing an actual mirrored sphere



[http://www.oakcorp.net/chaos/hdri.shtml]

Spherical Environment Mapping

[Mizutani, Reindel: Environment Mapping Algorithms, Reindel Software]

Spherical Environment Mapping

- to map the reflection vector to the sphere map, the following equations are used based on the reflection vector **r**
- in contrast to cube mapping, a generalized equation can be used

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{r_x}{2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} + \frac{1}{2} \\ \frac{r_y}{2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} + \frac{1}{2} \end{pmatrix}$$

Spherical Environment Mapping

- disadvantages
  - maps are hard to create on the fly
  - sampling is non-linear, non-uniform
  - sampling is view-point dependent
- advantages
  - no interpolation across map seems

Environment Mapping
Discussion

- object should be small
  compared to the environment
- issues with self-reflections
  and non-convex objects
- separate map for each object
- maps may need to be changed in case of a
  changing viewpoint due to non-uniform sampling
- translated objects might require a map update

Reflection - Summary

- planar reflections
  - generation of reflected geometry with reflected lighting
  - rendering of reflected geometry, reflection plane, original geometry
  - blending and stenciling is employed
- arbitrarily shaped reflectors
  - approximate reflections with environment mapping
  - cube maps
  - sphere maps
  - works best for distant environments without translation of objects
  - issues with sampling and concave objects