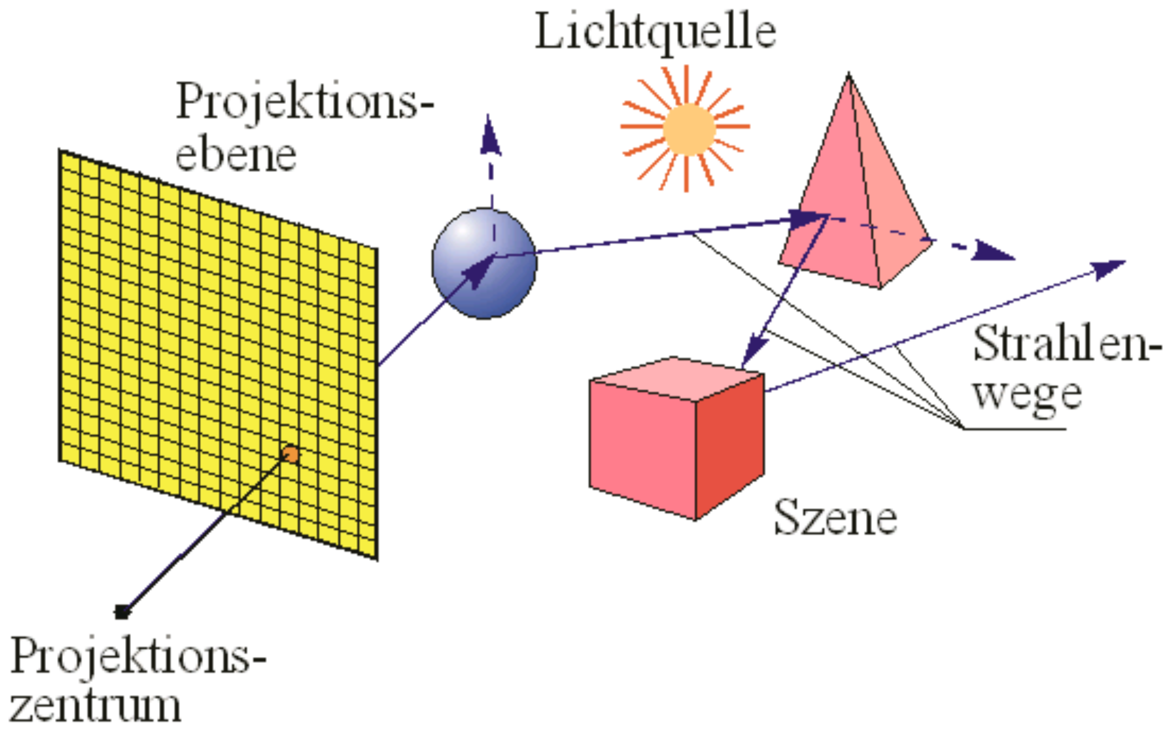
Visual Computing
University of Magdeburg

3. Rendering
3.8 Raytracing acceleration

3.8. Raytracing

- Concepts of Raytracing:

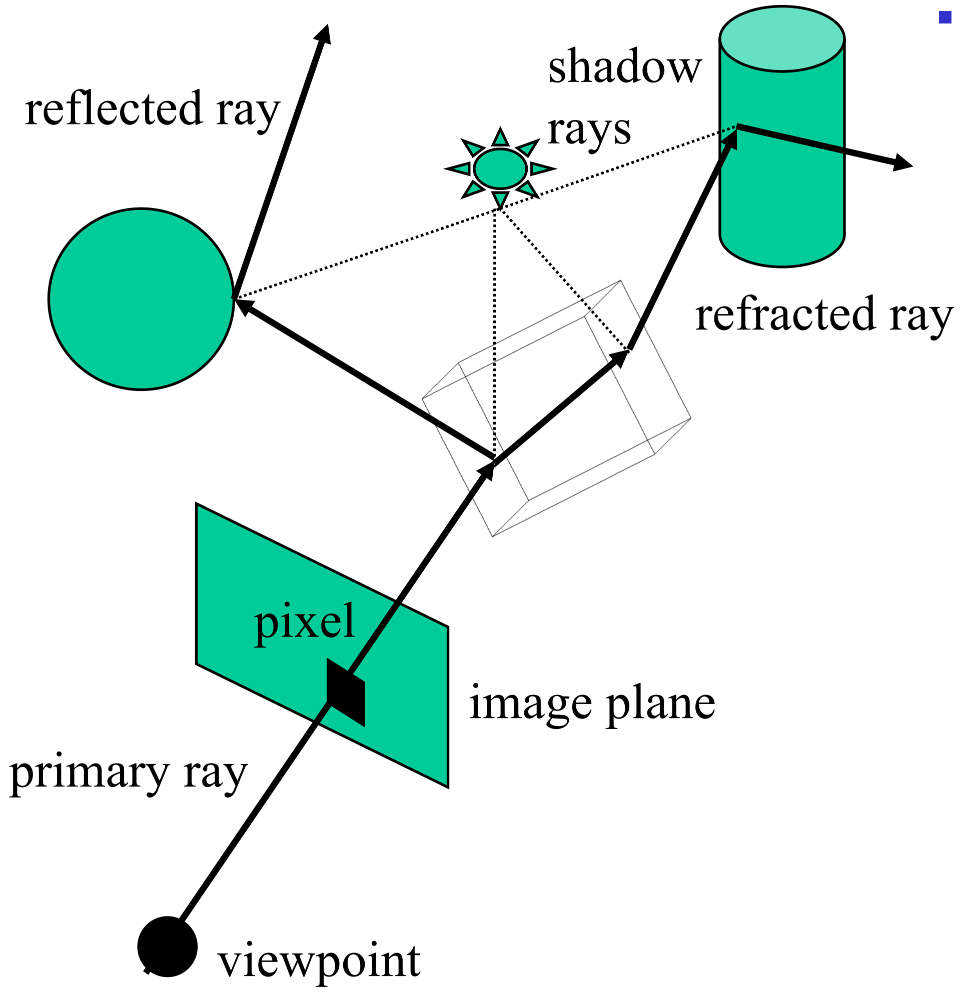
shadows
reflections
refractions
hidden surface removal
shading



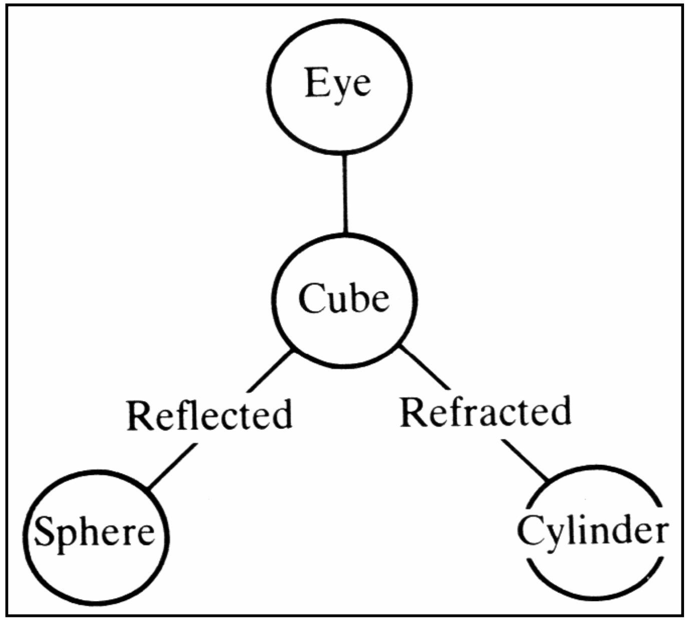
3.8. Raytracing

- **The Ray Tracing Algorithm**
  - One of the two fundamental rendering algorithms
  - Other one is *Rasterization*
- **Simple and intuitive**
  - Easy to understand and implement
- **Powerful and efficient**
  - Offers more features:
    shadows, reflection, refraction and other global effects
  - Efficient real-time implementation in SW and HW
- **Scalability**
  - Can work in parallel and distributed environments
  - Logarithmic scalability with scene size: O(log n) vs. O(n)

3.8. Raytracing



- **Recursive Ray Tracing**
- Searching recursively for paths to light sources
  - Interaction of light & material at intersection points
  - Recursively trace new rays in reflection, refraction and light direction



3.8. Raytracing

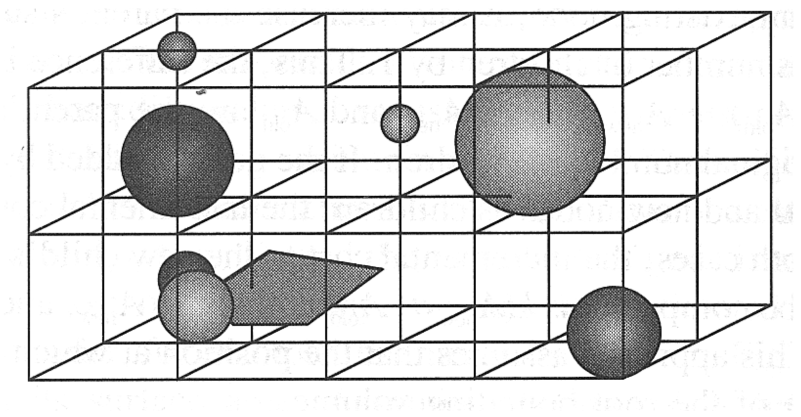
- **Ray Tracing Acceleration**
  - Intersect ray with all objects
    - Way too expensive
  - Faster intersection algorithms
    - Little effect

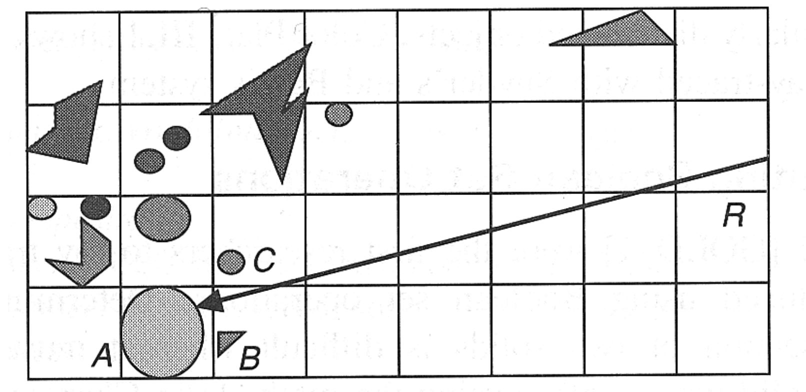
3.8. Raytracing

- **Ray Tracing Acceleration**
  - **Less intersection computations**
    - Space partitioning (often hierarchical)
      - Grid
      - Octree
      - Binary space partition (BSP) or kd-tree
      - Bounding volume hierarchy (BVH)
  - **Tracing of continuous bundles of rays**
    - Exploits coherence of neighboring rays
    - Cone tracing
    - Beam tracing

3.8. Raytracing

- Grid
  - Partitioning with equal, fixed sized „voxels“
- Building a grid structure
  - Partition the bounding box (bb)
  - Inserting objects
    - Trivial: insert into all voxels overlapping objects bounding box
    - Easily optimized
- Traversal
  - Iterate through all voxels in order as pierced by the ray
  - Compute intersection with objects in each voxel
  - Stop if intersection found in current voxel





3.8. Raytracing

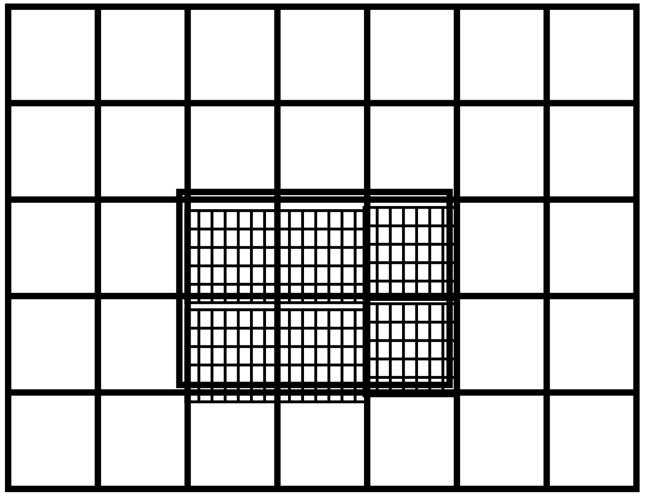
- **Grid: Issues**
  - **Grid traversal**
    - Requires enumeration of voxel along ray
    - Simple and hardware-friendly
  - **Grid resolution**
    - Strongly scene dependent
    - Cannot adapt to local density of objects
      - Problem: „Teapot in a stadium“
    - Possible solution: grids within grids – hierarchical grids

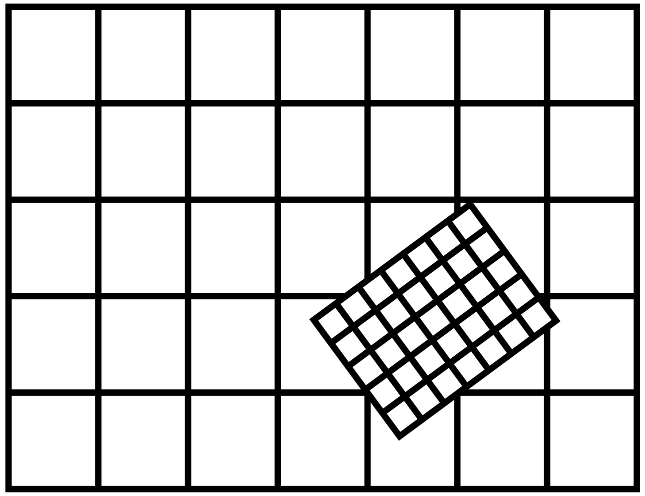
3.8. Raytracing

- **Grid: Issues**
  - Objects in multiple voxels
    - Store only references
    - Use mailboxing to avoid multiple intersection computations
      - Store (ray, object)-tuple in small cache (e.g. with hashing)
      - Do not intersect if found in cache

3.8. Raytracing

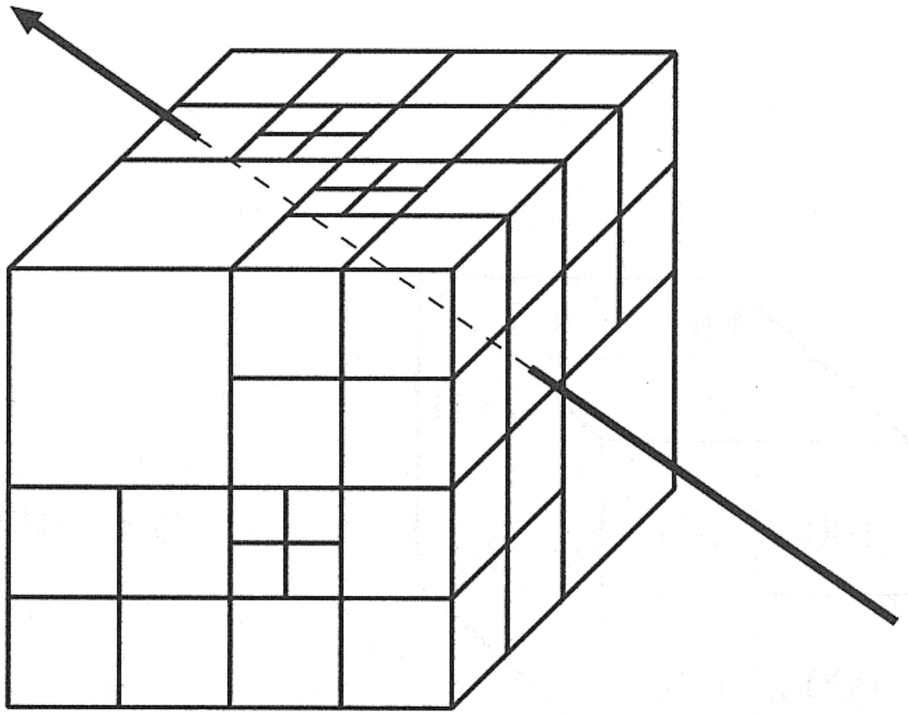
- **Hierarchical Grids**
  - **Simple building algorithm**
    - Coarse grid for entire scene
    - Recursively create grids in high-density voxels
    - Problem: What is the right resolution for each level?
  - **Advanced algorithm**
    - Place cluster of objects in separate grids
    - Insert these grids into parent grid
    - Problem: What are good clusters?





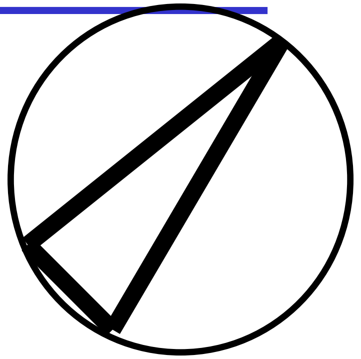
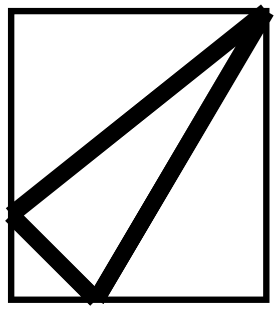
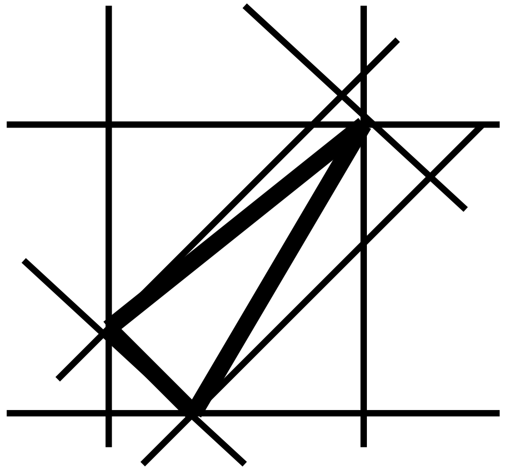
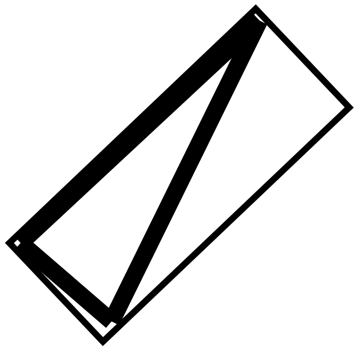
3.8. Raytracing

- **Octree**
  - **Hierarchical space partitioning**
    - Start with bounding box of entire scene
    - Recursively subdivide voxels into 8 equal sub-voxels
    - Criteria: Too many triangle & maxi
    - Result in adaptive subdivision
      - Allows for large traversal steps in empty regions
  - **Problems**
    - Pretty complex traversal algorithms
    - Slow to refine complex regions



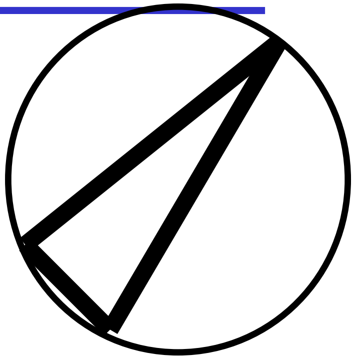
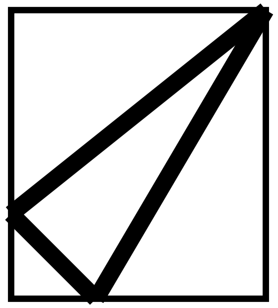
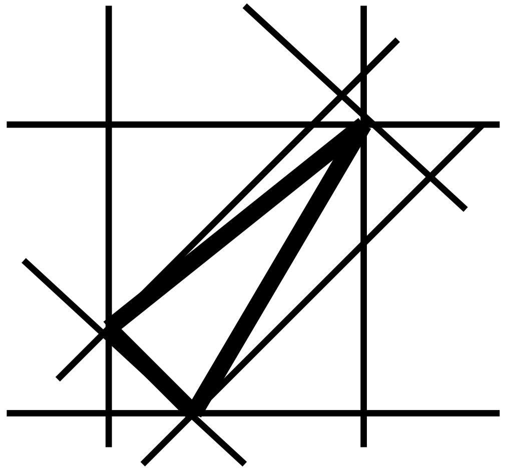
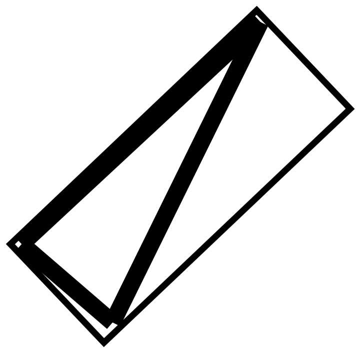
3.8. Raytracing

- **Bounding Volumes**
  - **Observation**
    - Bound geometry with BV
    - Only compute intersection if ray hits BV
  - **Sphere**
    - Very fast intersection computation
    - Often inefficient because too large
  - **Axis-aligned box**
    - Very simple computation (min-max)
    - Could be inefficient

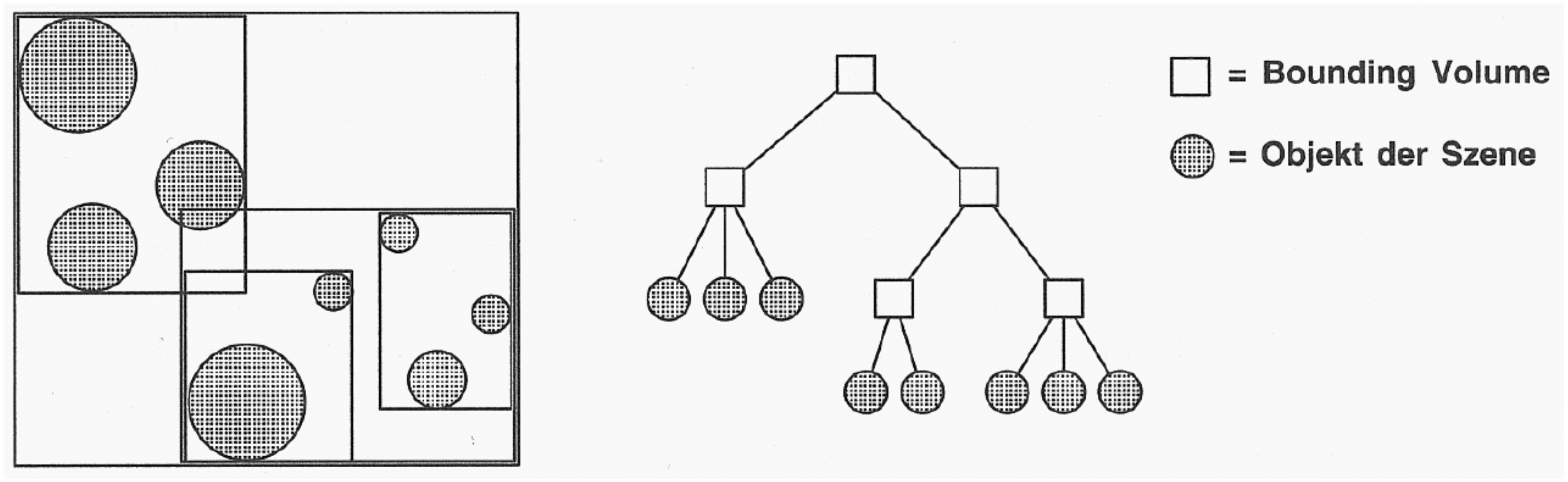
3.8. Raytracing

- **Bounding Volumes**
  - **Non-axis-aligned box**
    - Aka. „oriented bounding box (OBB)“
    - Often better fit
    - Fairly complex computation
  - **Slabs**
    - Pairs of half spaces
    - Fixed number of orientations
    - Fairly fast computation

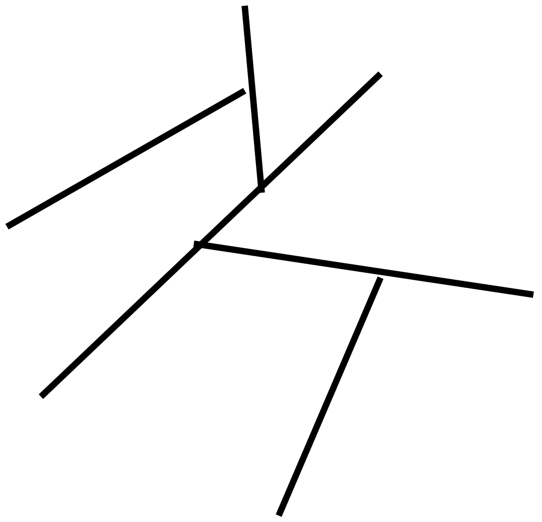
3.8. Raytracing

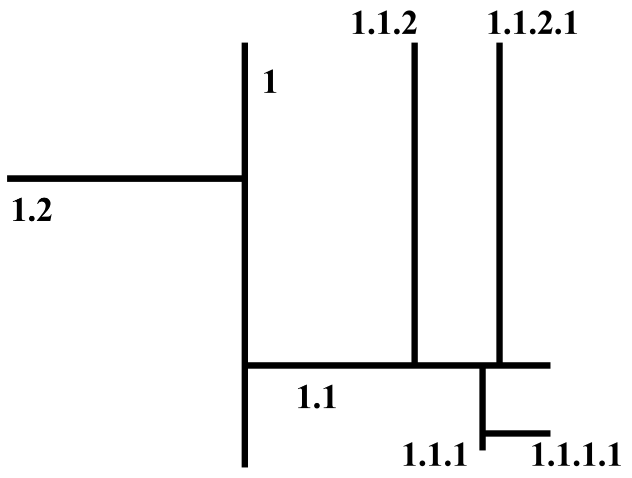
- **Bounding Volume Hierarchies**
  - **Idea**:
    - Organize BVs hierarchically into new BVs
  - **Advantages**:
    - Very good adaptivity
    - Efficient traversal O(log N)
  - **Problems**
    - How to arrange BVs?



3.8. Raytracing

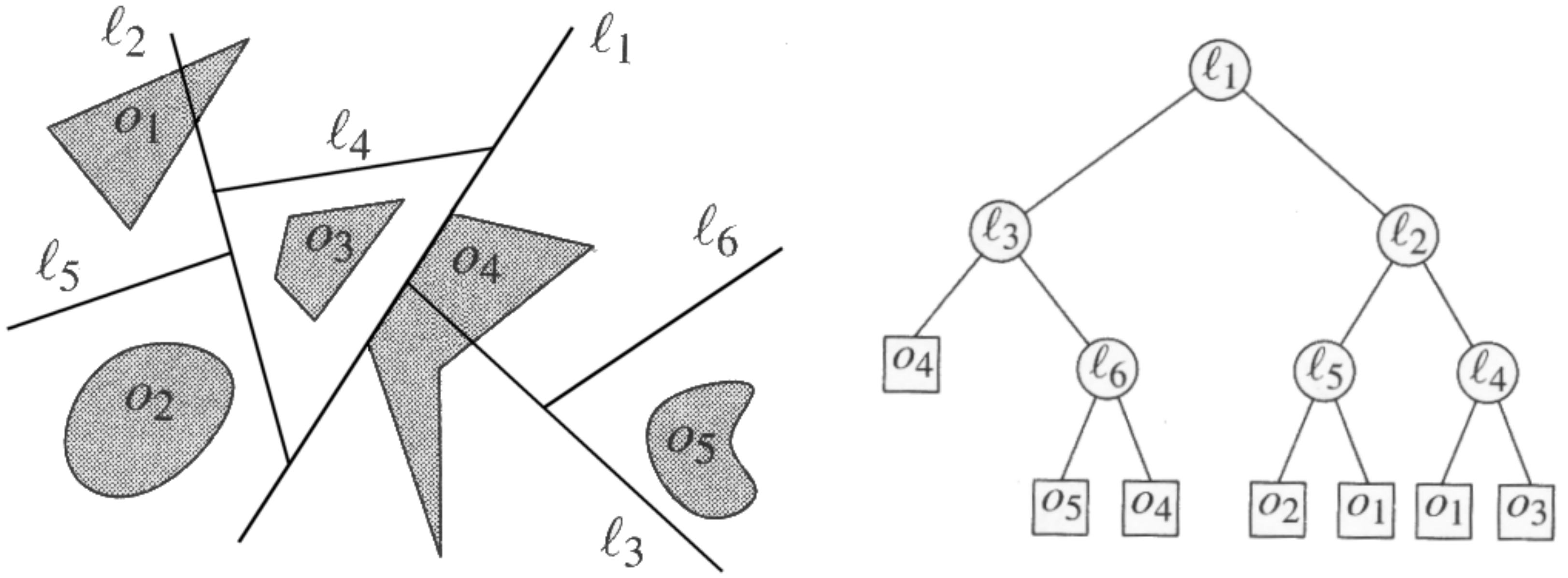
- BSP and kd-tree
  - Recursive space partitioning with half-spaces
    - **Binary Space Partition (BSP):**
      - Recursively split space into halves
      - Splitting with half-spaces in arbitrary position
        - Often defined by existing polygons
      - Often used for visibility in games
      - Traverse binary tree from front to back
    - **kd-tree**
      - Special case of BSP
        - Splitting with axis-aligned half-spaces
      - Defined recursively through nodes with
        - Axis-flag
        - Split location (1D)
        - Child pointer(s)





BSP

Use planes to recursively split our object space, keeping a tree structure of these recursive splits.

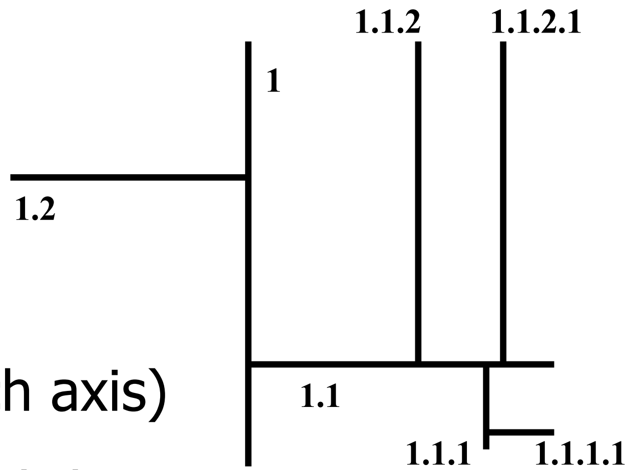


3.8. Raytracing

- **Querying the BSP Tree**
  - Place eye point into the tree
  - Traverse tree from root
  - Compute first intersection ray-object for objects in the half space where the eye point is not located, then with all objects on the intersecting plane, then all objects on the half space containing the eye point

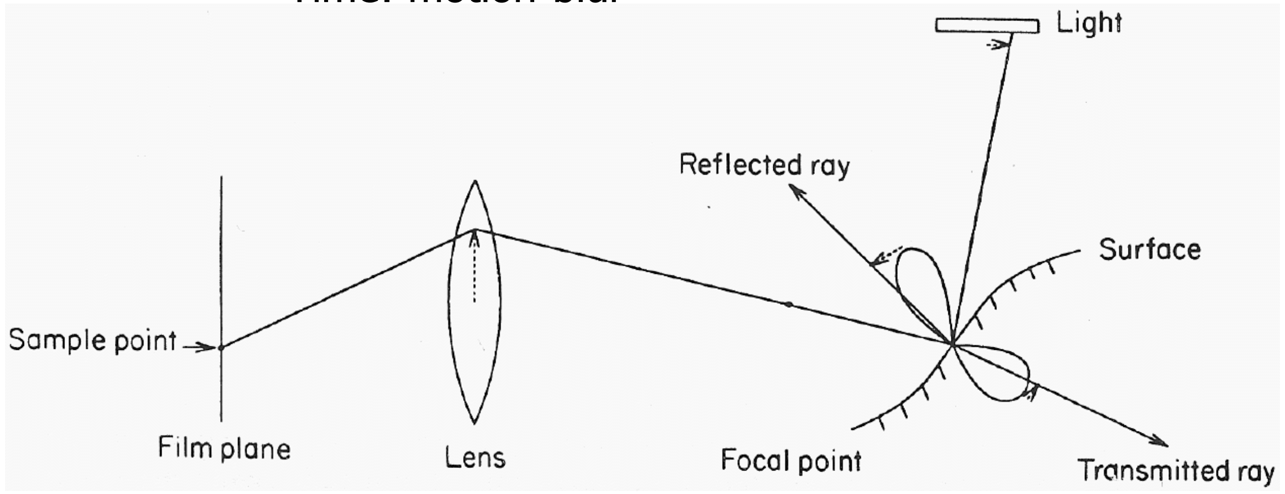
3.8. Raytracing

- **Kd-Tree**
  - **Build procedure**
    - Select bounding box of scene as root node
    - Select a best split plane location (along each axis)
      - Global optimum is NP-hard → use local heuristics
      - General criteria: Cut away large and simple regions (e.g. empty)
    - Recurse, unless child
      - Contains only a few objects (2-3 typically)
      - Depth become too large (20-30, depending on scene)
      - Optimization: Add nodes to cut away empty space in leaf nodes



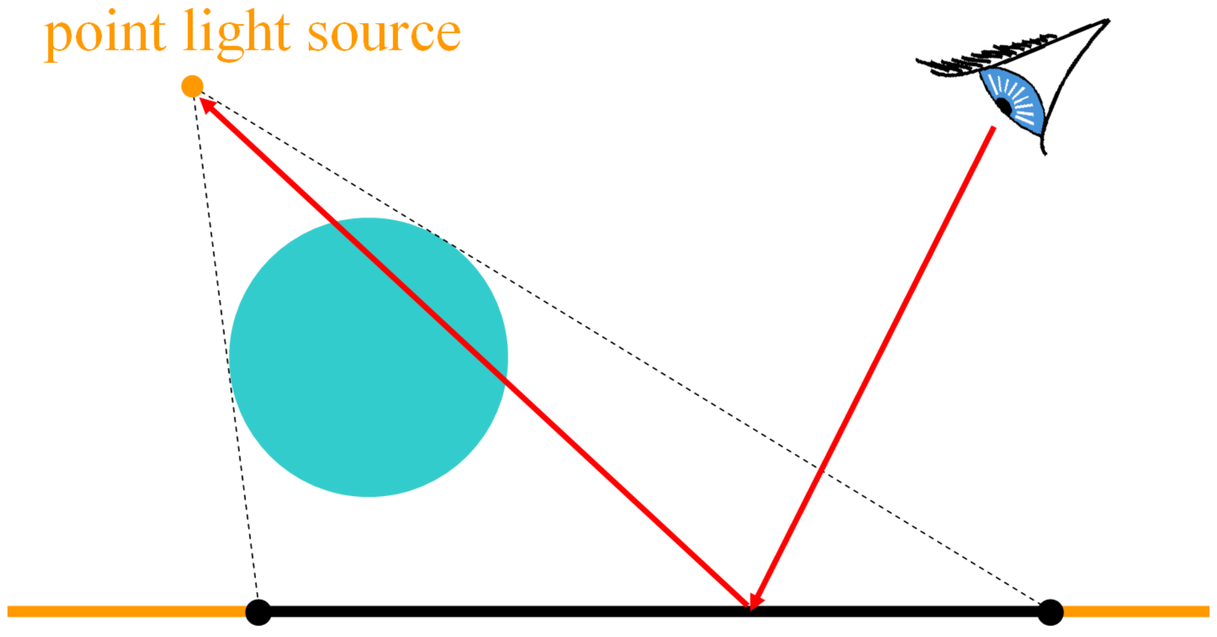
3.8. Raytracing

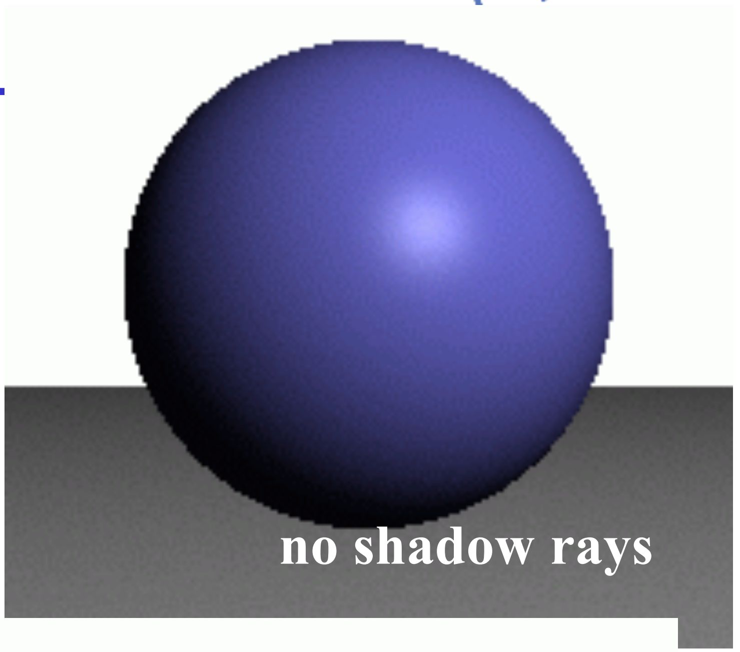
- **Distribution Ray-Tracing**
  - Formerly called Distributed Ray-Tracing [Cook`84]
  - Stochastic sampling
    - Illumination: extended light sources; soft shadows
    - Pixel: Anti-Aliasing
    - Lens: Depth-of-Field
    - BRDF: Glossy reflections
    - Time: motion-blur

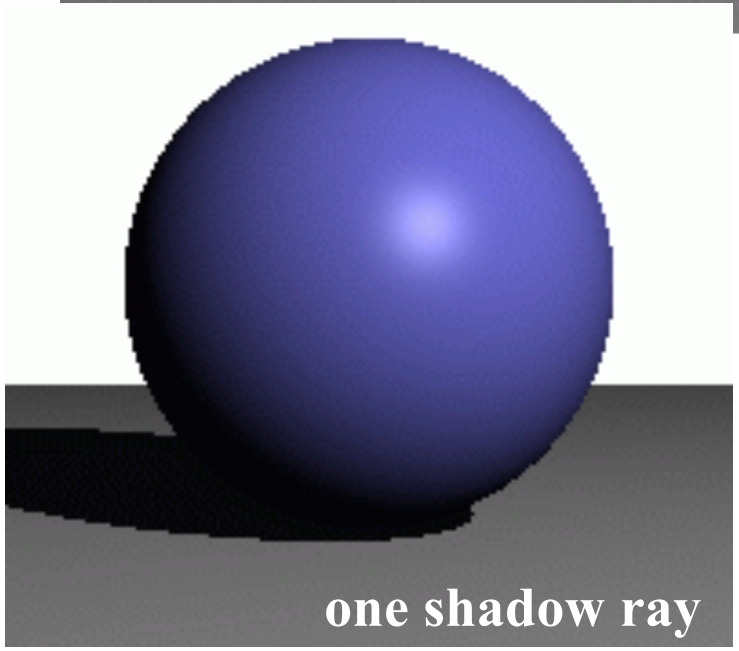


3.8. Raytracing

- **Distribution Ray-Tracing - Shadows**
- one shadow ray per intersection per point light source

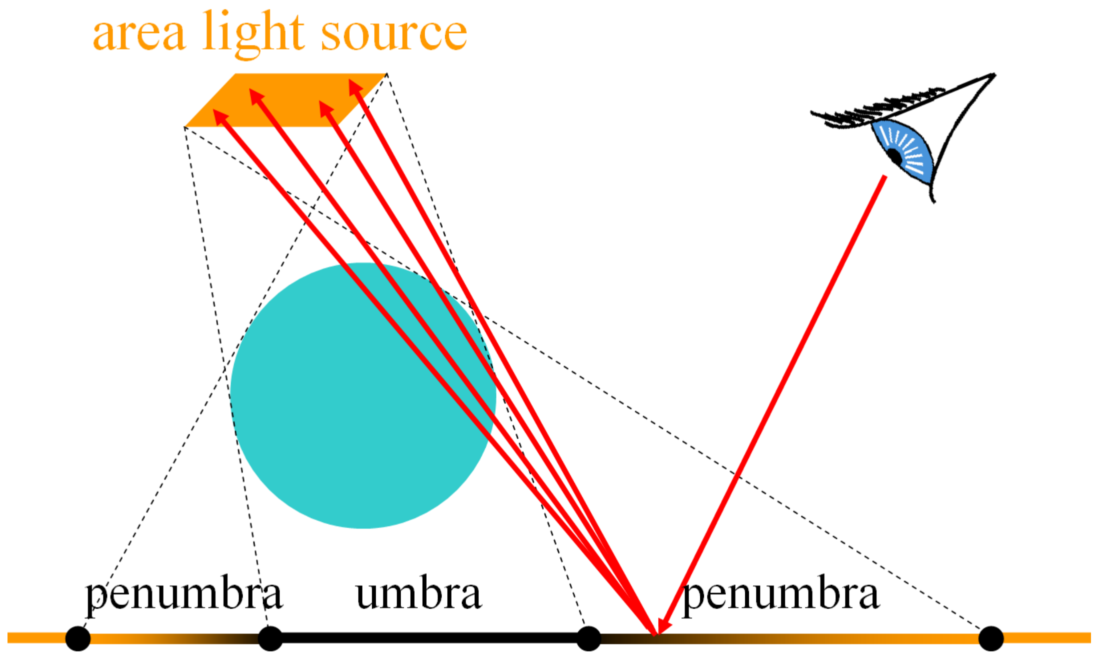


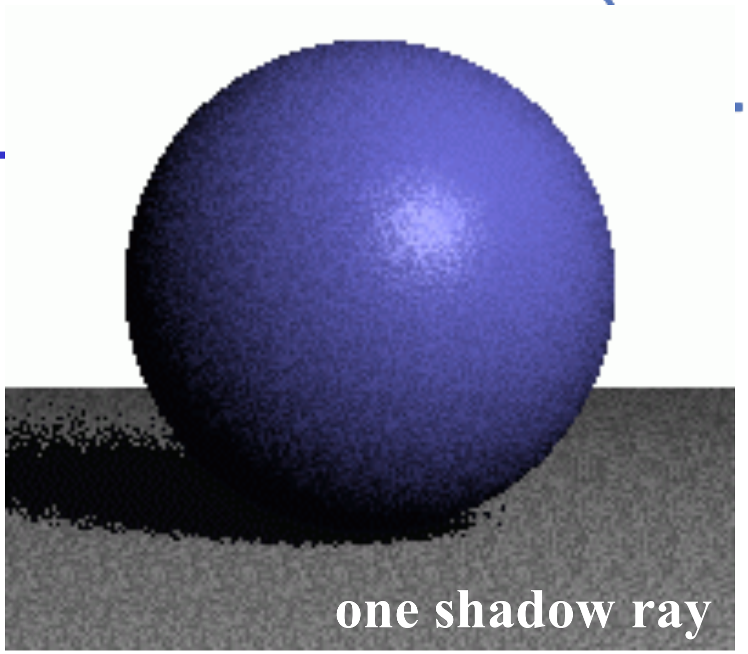


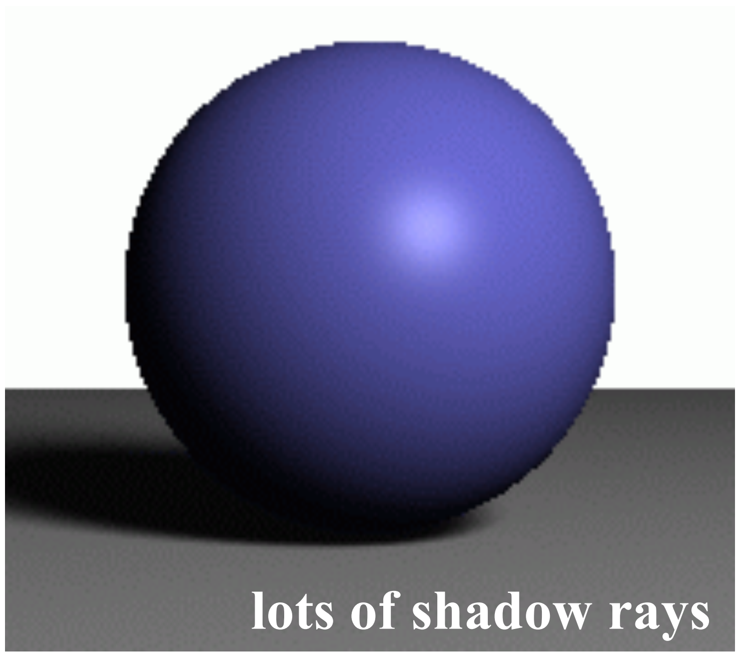


3.8. Raytracing

- **Distribution Ray-Tracing - Shadows**
- multiple shadow rays to sample area light source

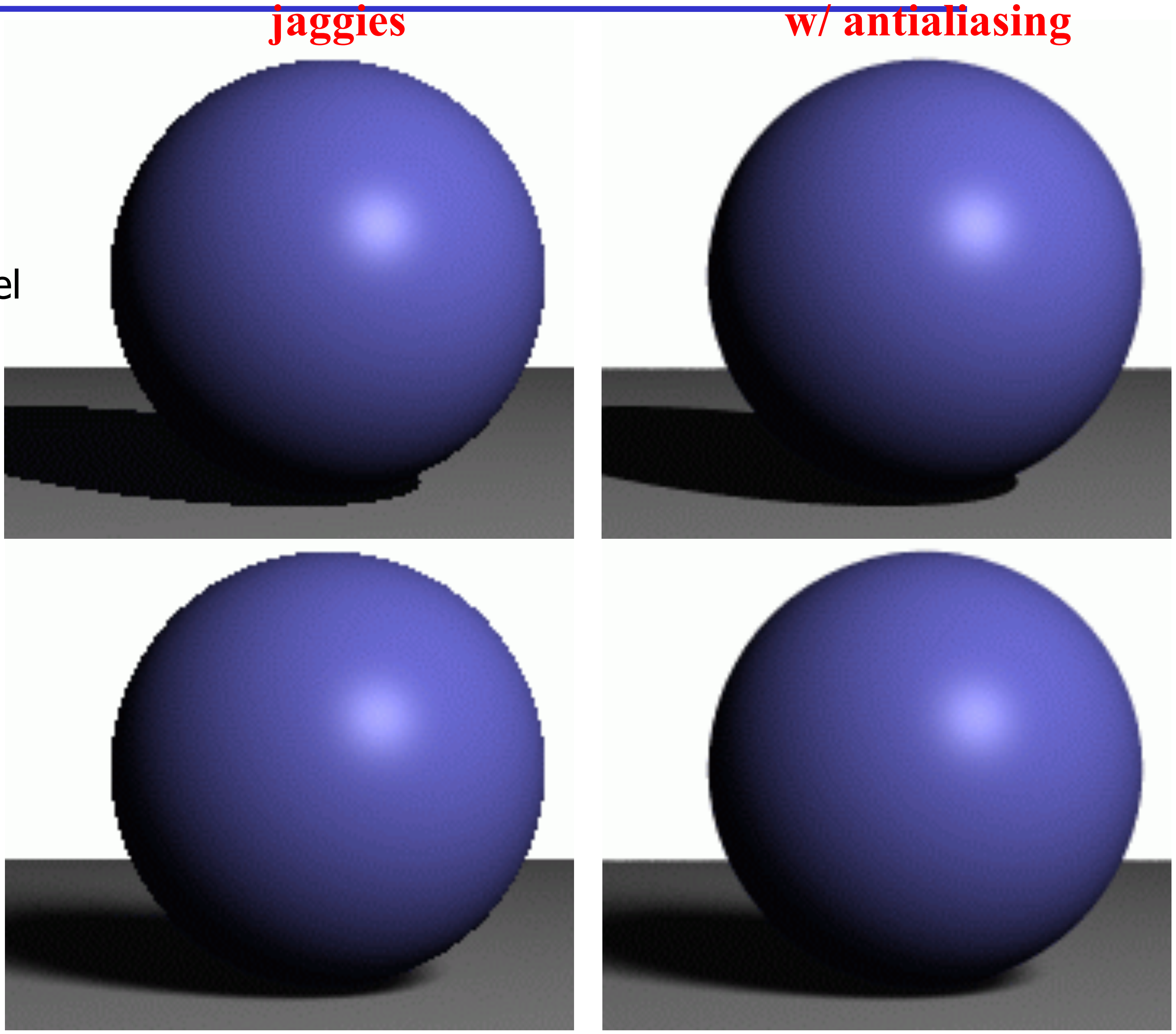






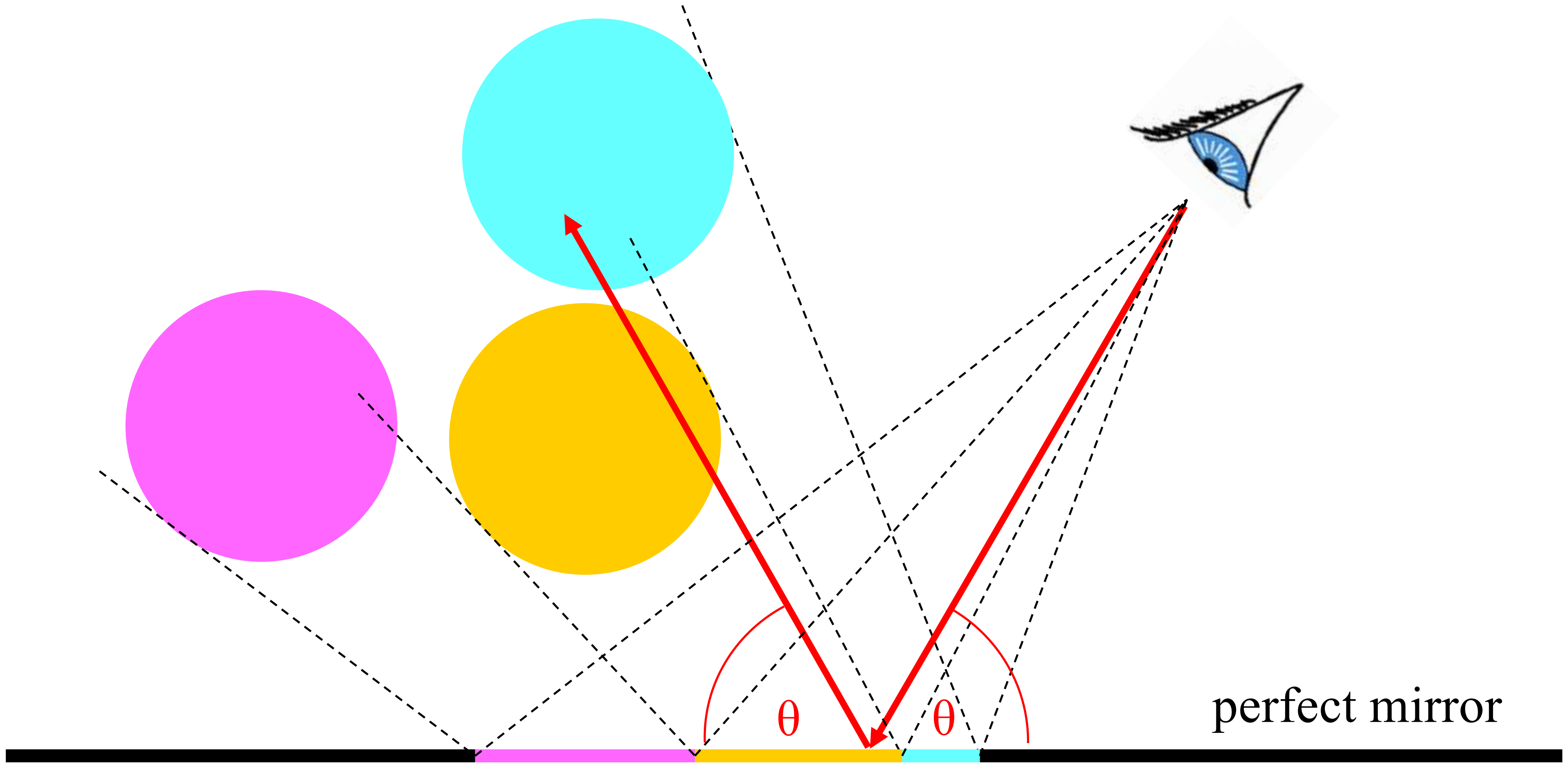
3.8. Raytracing

- **Distribution Ray-Tracing - Antialiasing – Supersampling**
- multiple rays per pixel



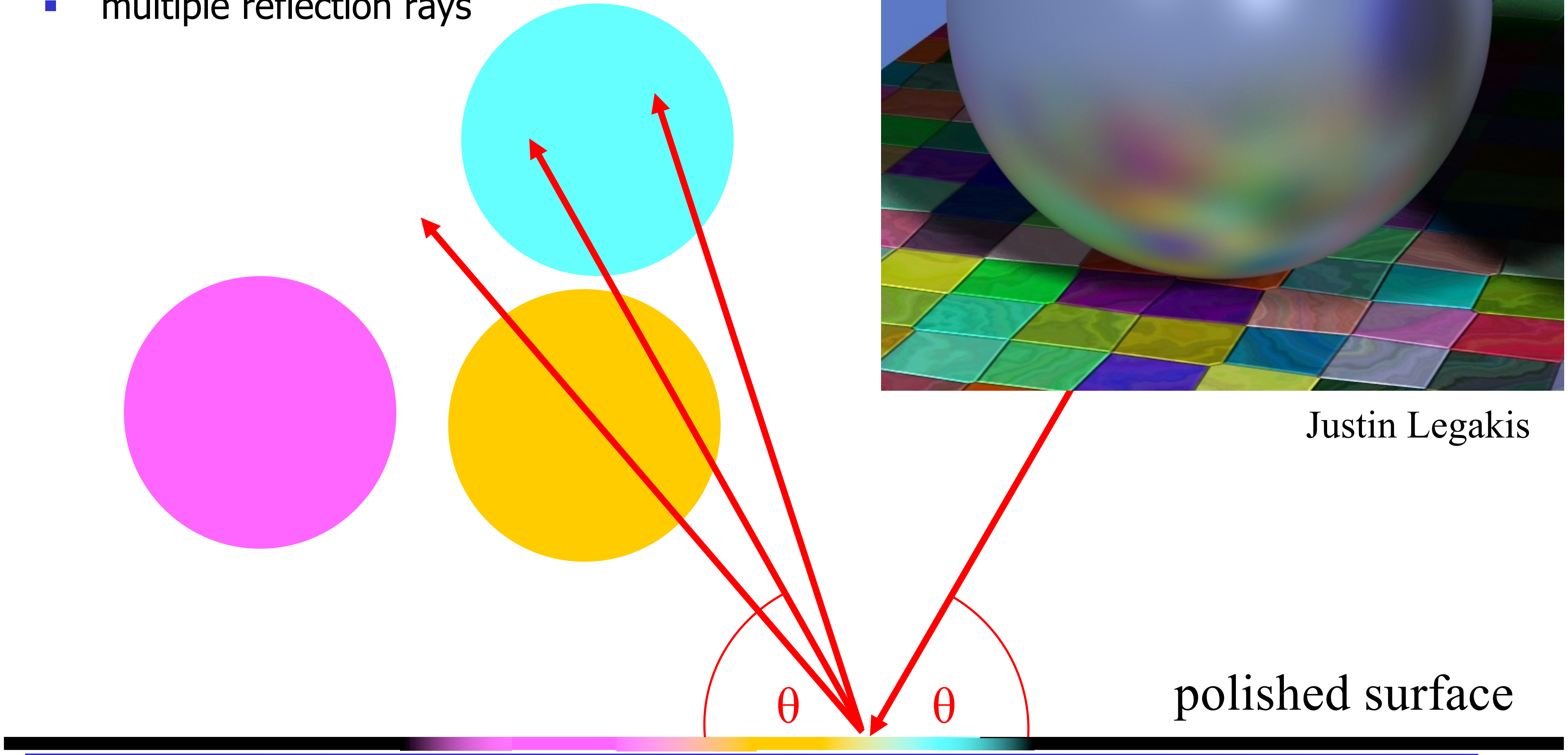
3.8. Raytracing

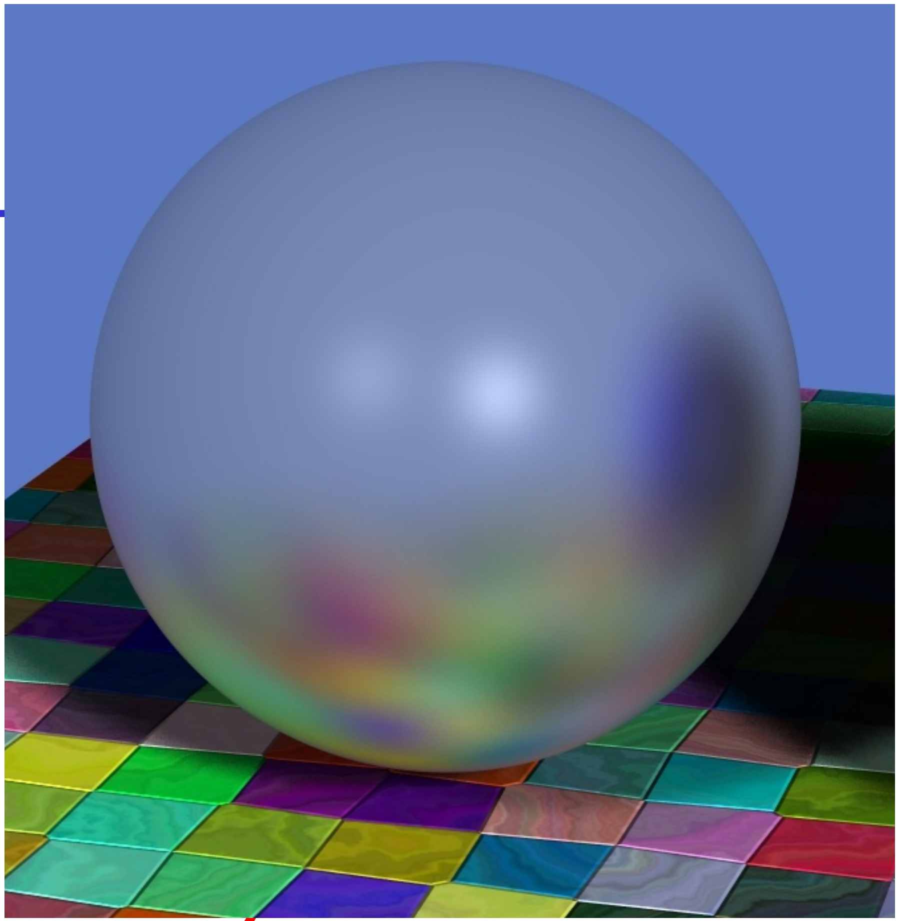
- **Distribution Ray-Tracing - Reflection**
- one reflection ray per intersection



3.8. Raytracing

- **Distribution Ray-Tracing – Glossy Reflection**
- multiple reflection rays

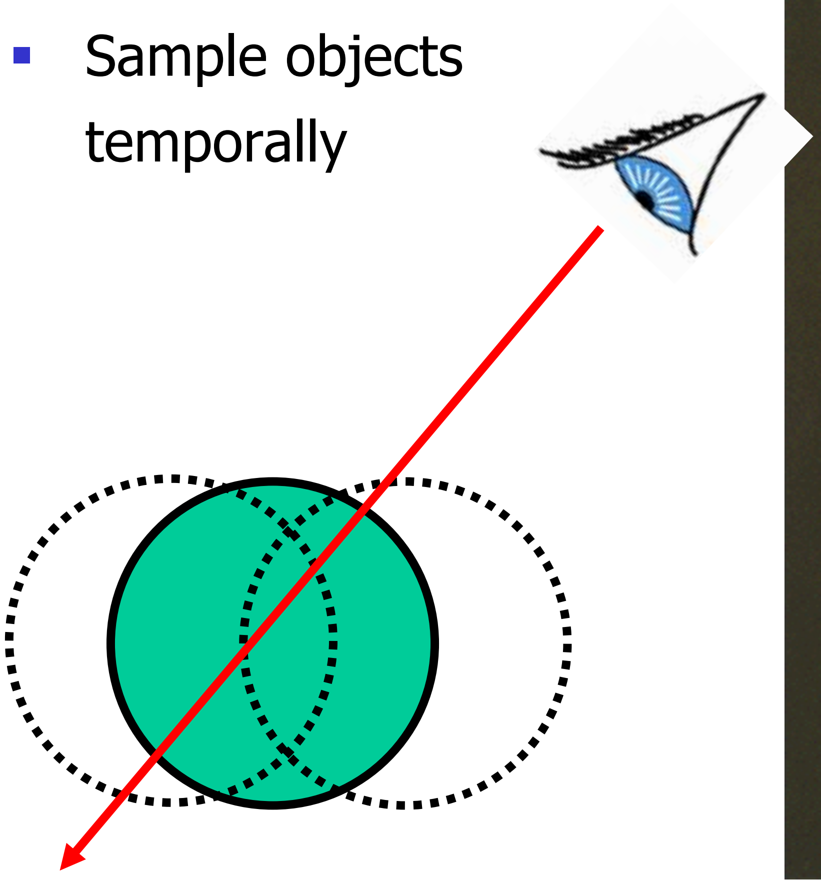


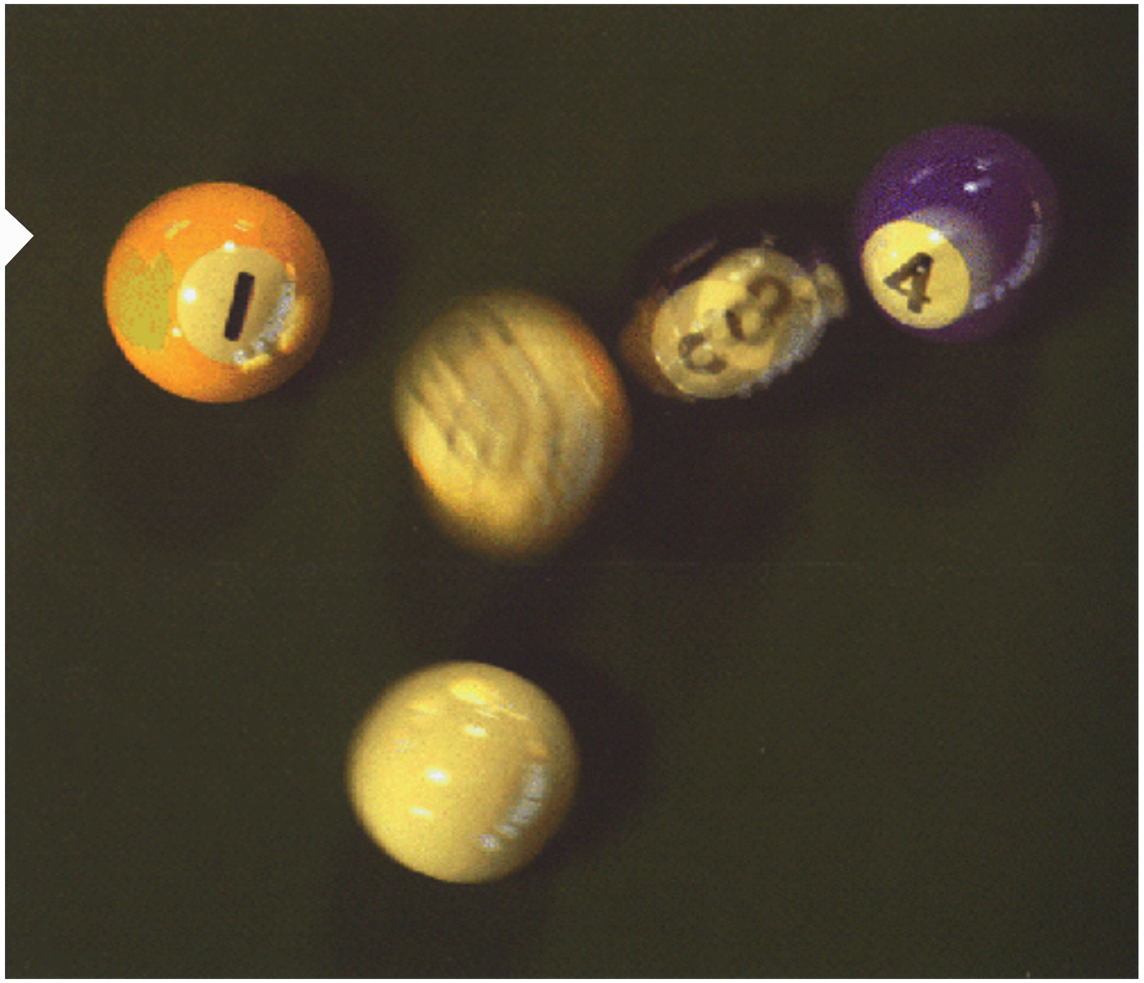


Justin Legakis

3.8. Raytracing

- **Distribution Ray-Tracing - Motion Blur**
- Sample objects temporally

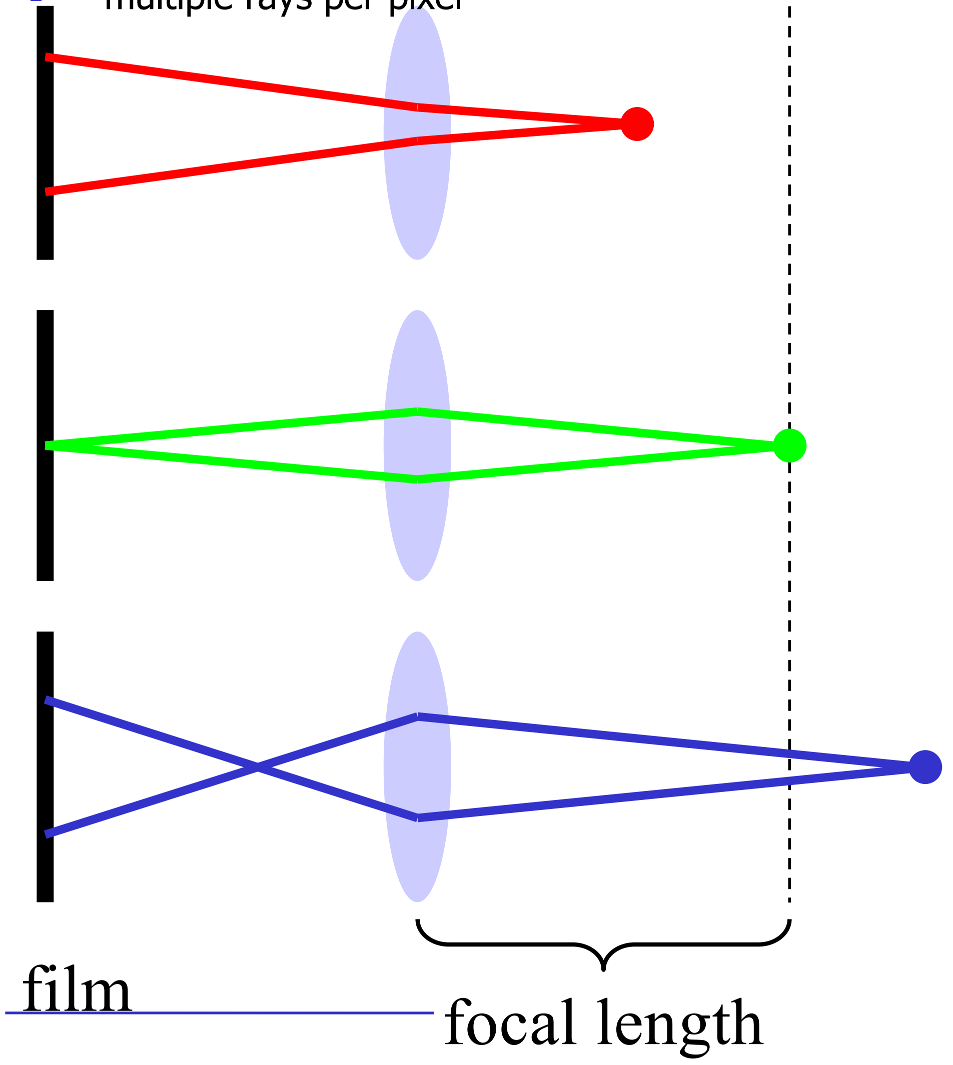


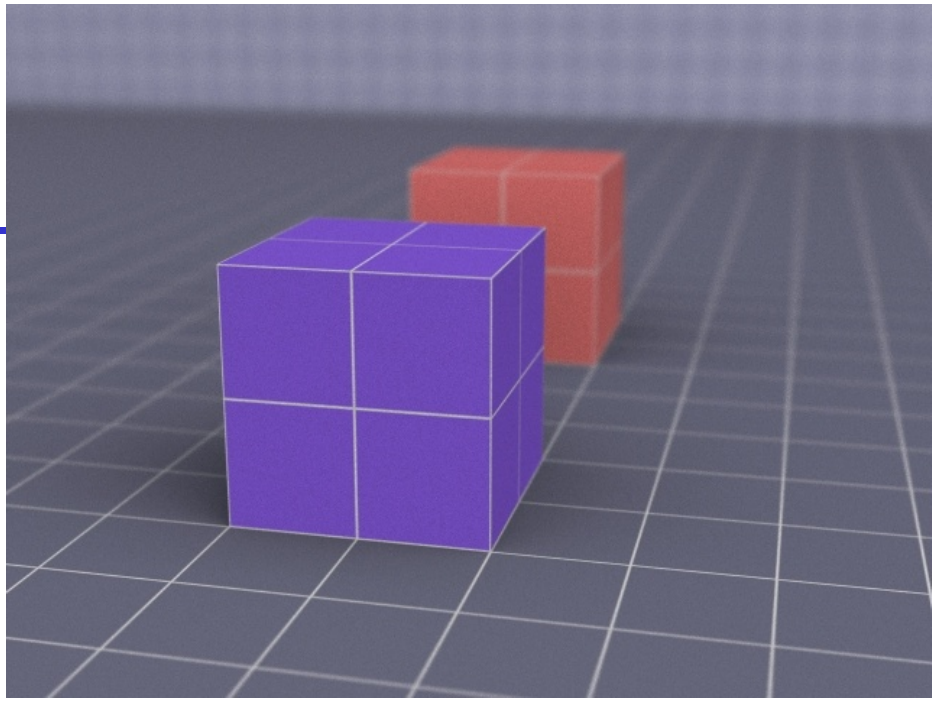


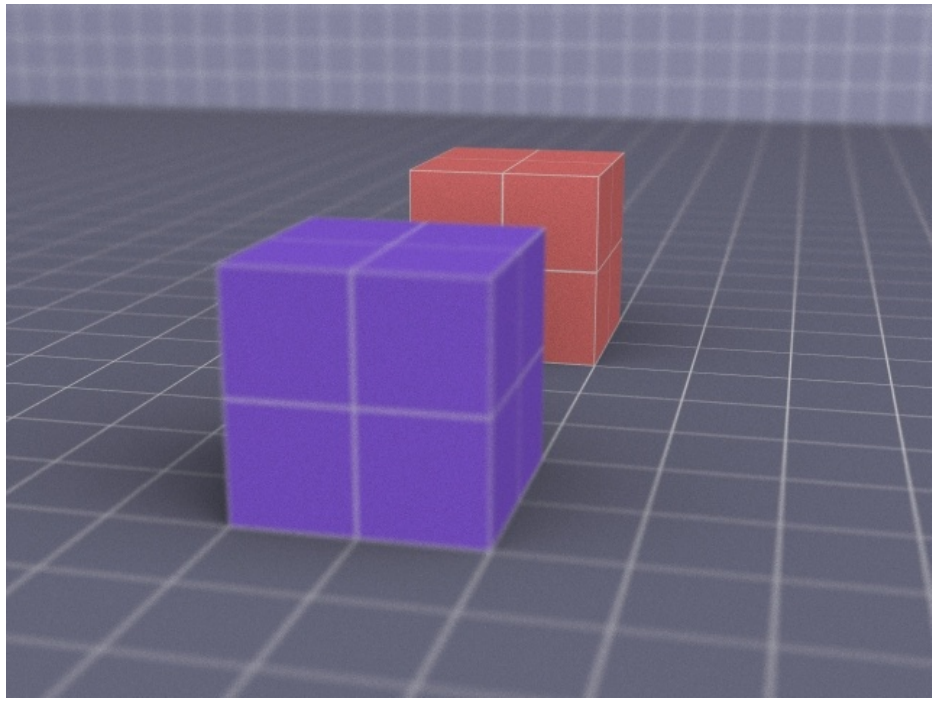
Rob Cook

3.8. Raytracing

- **Distribution Ray-Tracing – Depth of Field**
- multiple rays per pixel

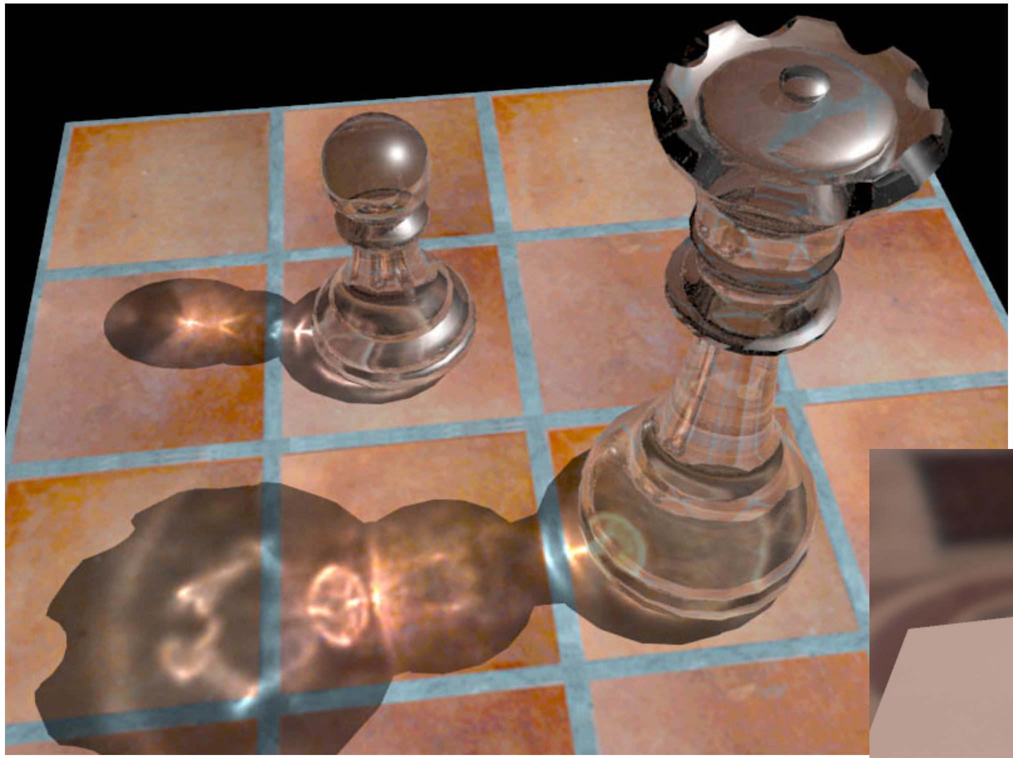
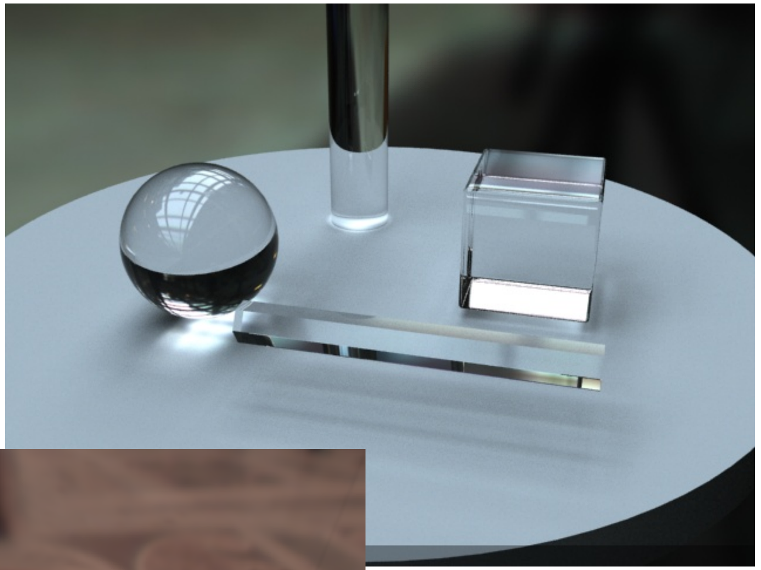






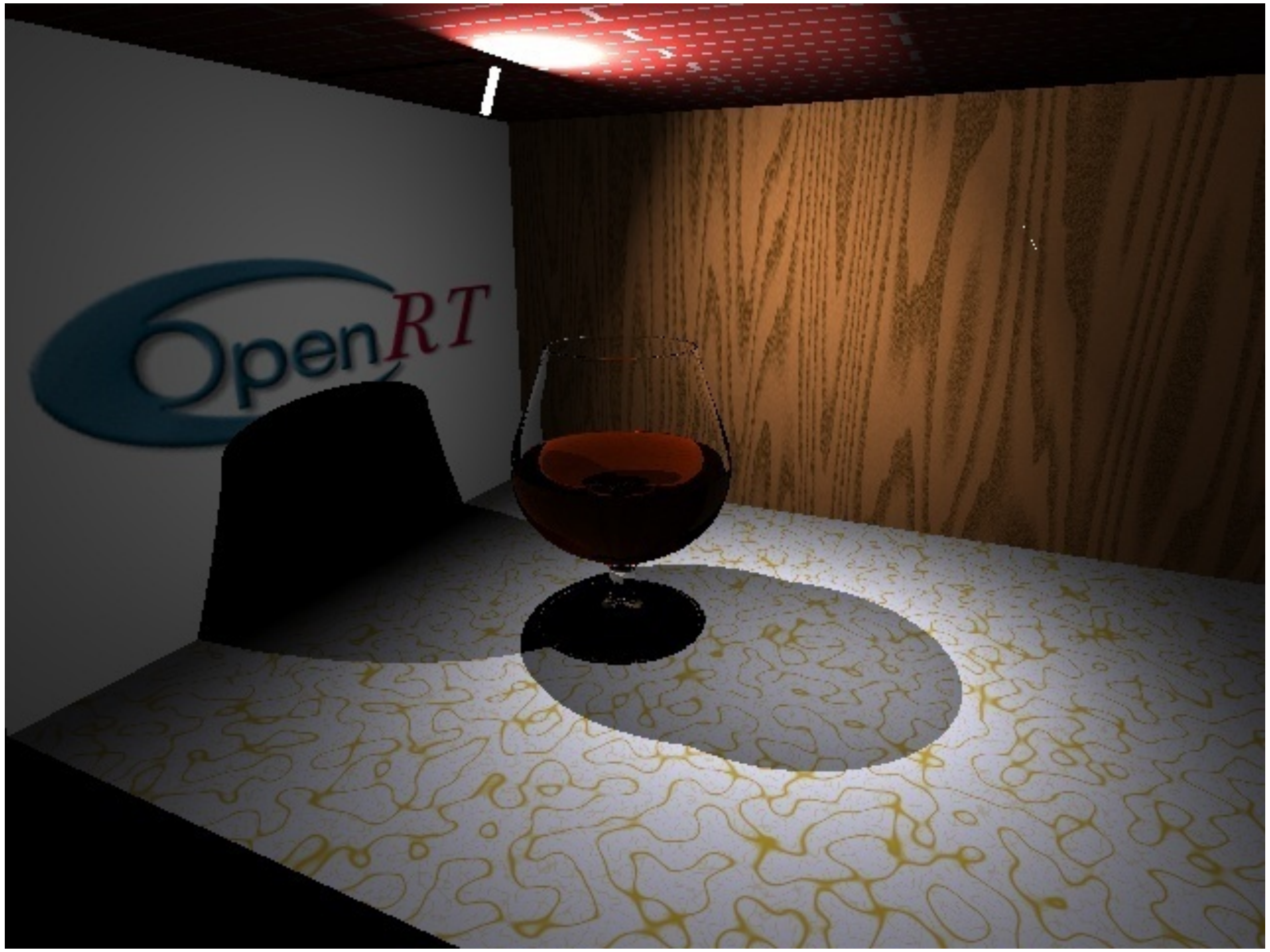
3.8. Raytracing

- Photone Mapping: forward raytracing (ie. sending rays from light sources) calculate reflecting and refracting light (aka. caustics).

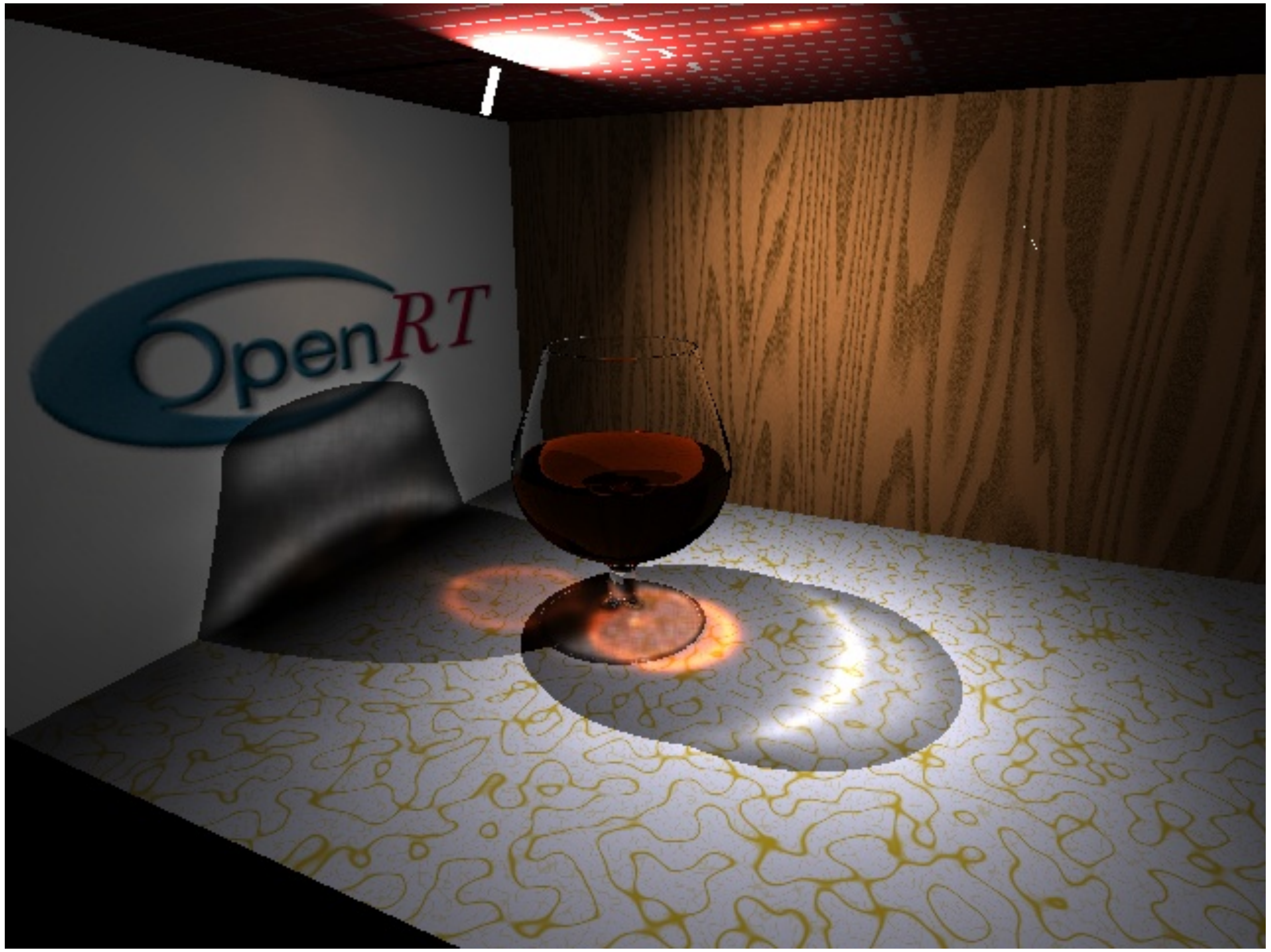
3.8. Raytracing

- **Photone Mapping**
- What is wrong with this picture?



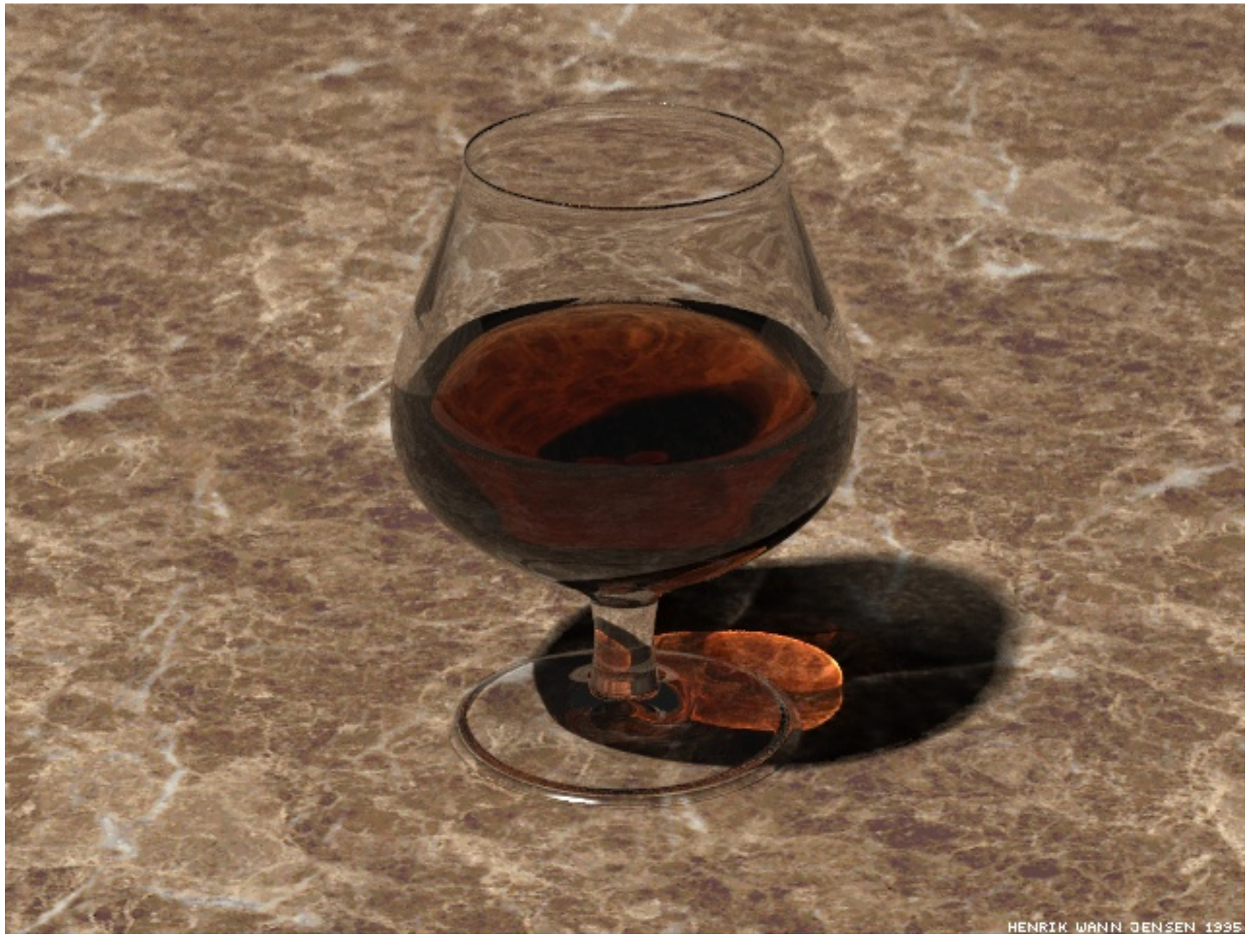
3.8. Raytracing

- **Photone Mapping**
- With caustic illumination



3.8. Raytracing

- Photone Mapping



3.8. Raytracing

- **Wrap-Up**
  - **Recursive Ray Tracing**
    - Secondary rays
      - Reflection
      - Refraction
      - To light sources
  - **Acceleration techniques**
    - Grid decomposition
    - Hierarchical Trees
    - Octree
    - Bounding volume hierarchies
    - BSP tree
    - Kd-Tree
  - **Distribution Ray Tracing**