Ray Tracing II

These slides are not so exciting, we skiped over most

Based on slides by Steve Marschner

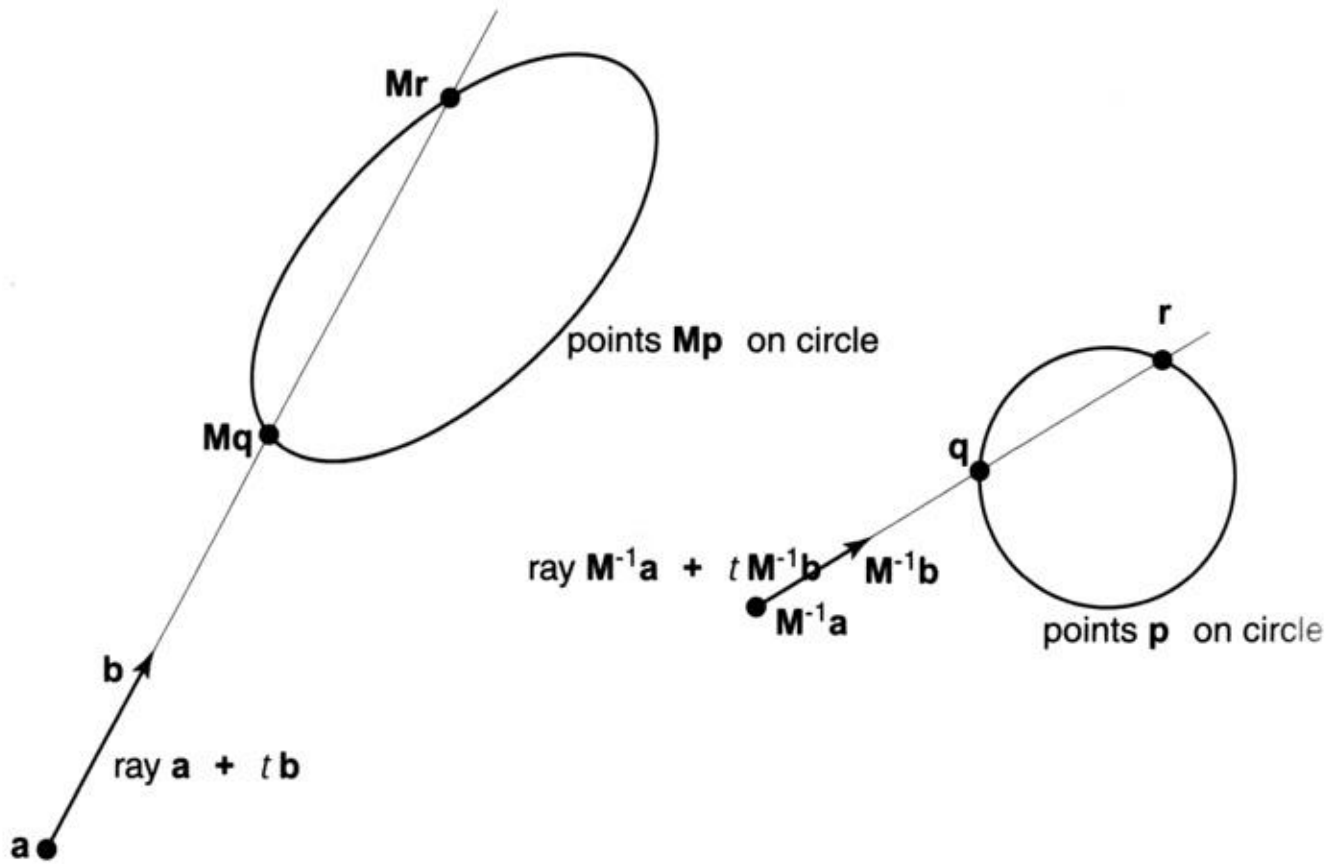
Topics

- Transformations in ray tracing
  - Transforming objects
  - Transformation hierarchies
- Ray tracing acceleration structures
  - Bounding volumes
  - Bounding volume hierarchies
  - Uniform spatial subdivision
  - Adaptive spatial subdivision

Transforming objects

- In modeling, we've seen the usefulness of transformations
  - How to do the same in RT?
- Take spheres as an example: want to support transformed spheres
  - Need a new Surface subclass
- Option 1: transform sphere into world coordinates
  - Write code to intersect arbitrary ellipsoids
- Option 2: transform ray into sphere's coordinates
  - Then just use existing sphere intersection routine

Intersecting transformed objects



[Shirley 2000]

Based on slides by Steve Marschner

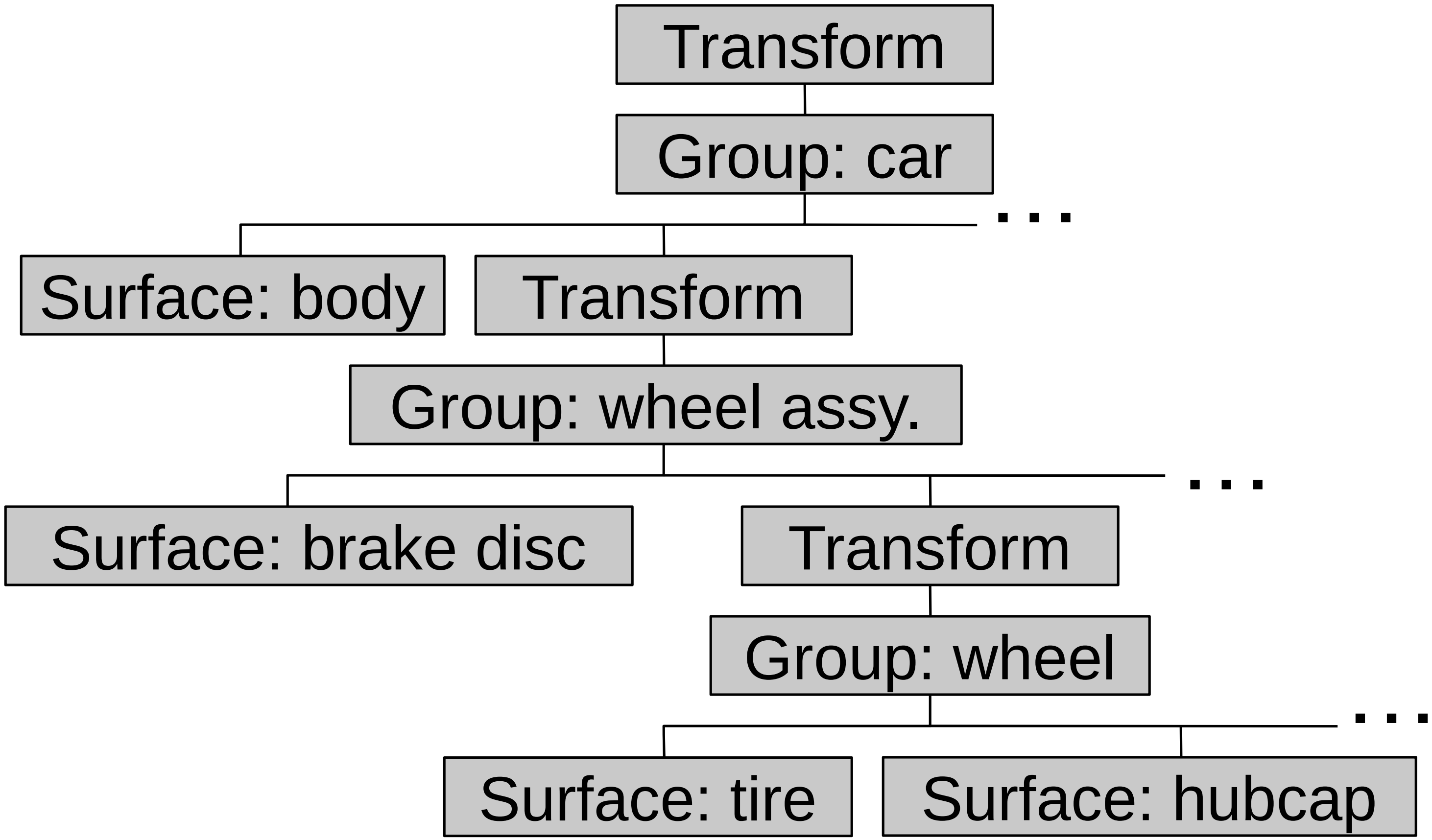
Implementing RT transforms

- Create wrapper object "TrasformedSurface"
  - Has a transform T and a reference to a surface S
  - To intersect:
    - Transform ray to local coords (by inverse of T)
    - Call surface.intersect
    - Transform hit data back to global coords (by T)
      - Intersection point
      - Surface normal
      - Any other relevant data (maybe none)

Groups, transforms, hierarchies

- Often it’s useful to transform several objects at once
  - Add “SurfaceGroup” as a subclass of Surface
    - Has a list of surfaces
    - Returns closest intersection
      - Opportunity to move ray intersection code here to avoid duplication
- With TransformedSurface and SurfaceGroup you can put transforms below transforms
  - This gives us a transformation hierarchy.

A transformation hierarchy

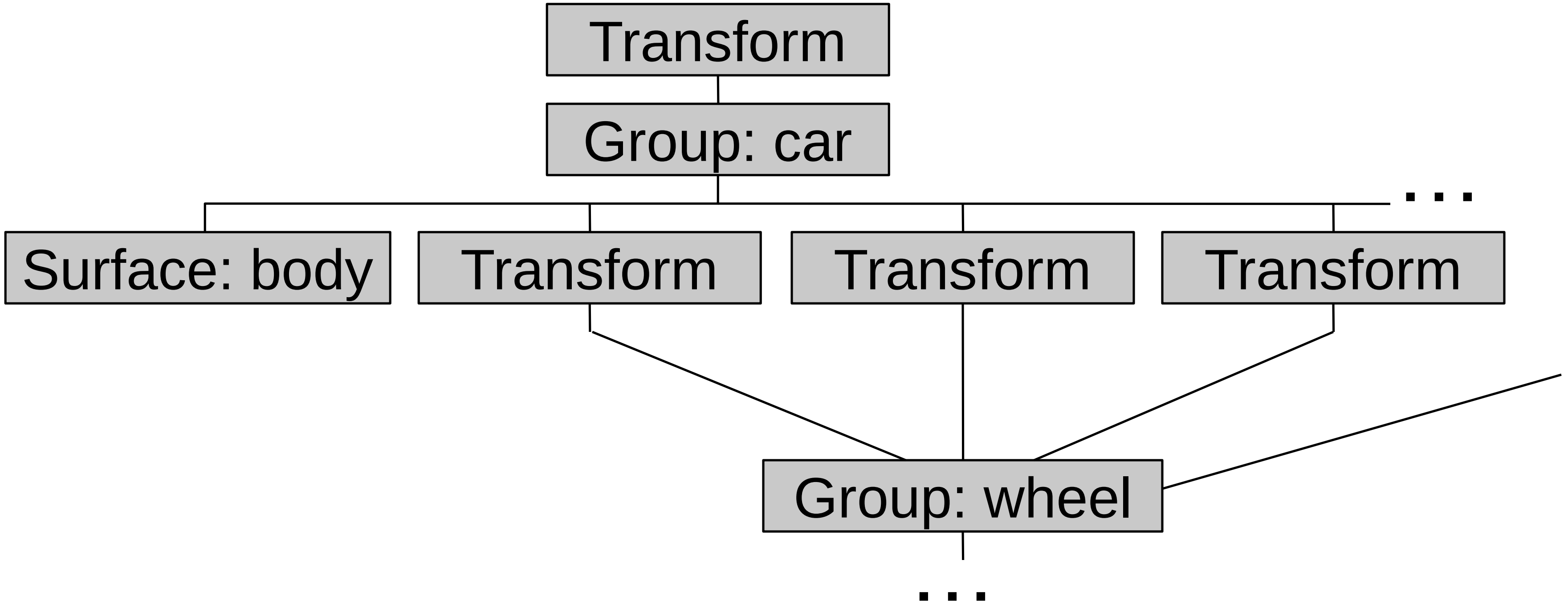


- Common optimization: merge transforms with groups
  - This is how we did it in the modeler assignment

Instancing

- Anything worth doing is worth doing *n* times
- If we can transform objects, why not transform them several ways?
  - Many models have repeated subassemblies
    - Mechanical parts (wheels of car)
    - Multiple objects (chairs in classroom, …)
  - Nothing stops you from creating two TransformedSurface objects that reference the same Surface
    - Allowing this makes the transformation tree into a DAG
      - (directed acyclic graph)
    - Mostly this is transparent to the renderer

Hierarchy with instancing



Based on slides by Steve Marschner

Hierarchies and performance

- Transforming rays is expensive
  - minimize tree depth: flatten on input
    - push all transformations toward leaves
    - triangle meshes may do best to stay as group
      - transform ray once, intersect with mesh
  - internal group nodes still required for instancing
    - can’t push two transforms down to same child!

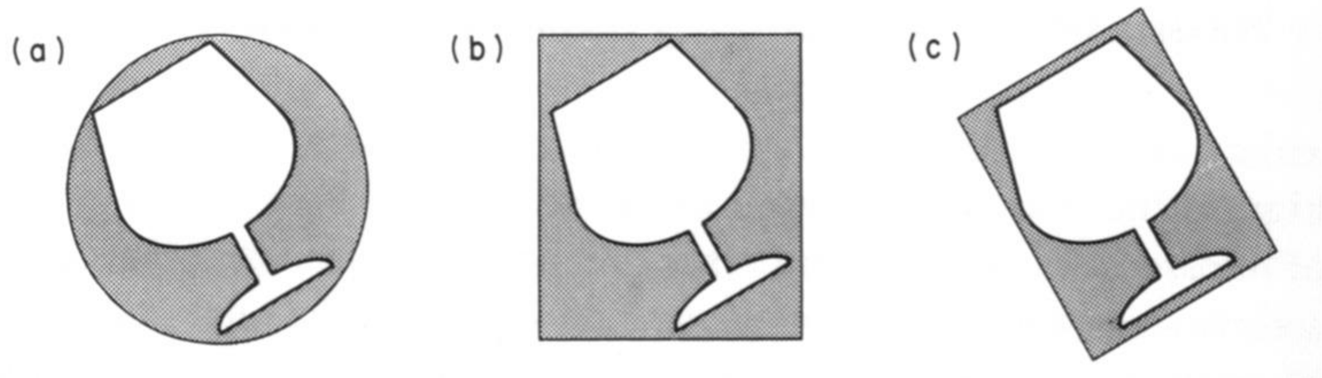
Based on slides by Steve Marschner

Ray tracing acceleration

- Ray tracing is slow. This is bad!
  - Ray tracers spend most of their time in ray-surface intersection methods
- Ways to improve speed
  - Make intersection methods more efficient
    - Yes, good idea. But only gets you so far
  - Call intersection methods fewer times
    - Intersecting every ray with every object is wasteful
    - Basic strategy: efficiently find big chunks of geometry that definitely do not intersect a ray

Bounding volumes

- Quick way to avoid intersections: bound object with a simple volume
  - Object is fully contained in the volume
  - If it doesn't hit the volume, it doesn't hit the object
  - So test bvol first, then test object if it hits



[Glassner 89, Fig 4.5]

Based on slides by Steve Marschner

Bounding volumes

- Cost: slightly more for hits and near misses, much less for far misses
- Worth doing? It depends:
  - Cost of bvol intersection test should be small
    - Therefore use simple shapes (spheres, boxes, …)
  - Cost of object intersect test should be large
    - Bvols most useful for complex objects
  - Tightness of fit should be good
    - Loose fit leads to extra object intersections
    - Tradeoff between tightness and bvol intersection cost

Implementing bounding volume

- Just add new Surface subclass, “BoundedSurface”
  - Contains a bounding volume and a reference to a surface
  - Intersection method:
    - Intersect with bvol, return false for miss
    - Return surface.intersect(ray)
  - Like transformations, common to merge with group
  - This change is transparent to the renderer (only it might run faster)
- Note that all Surfaces will need to be able to supply bounding volumes for themselves

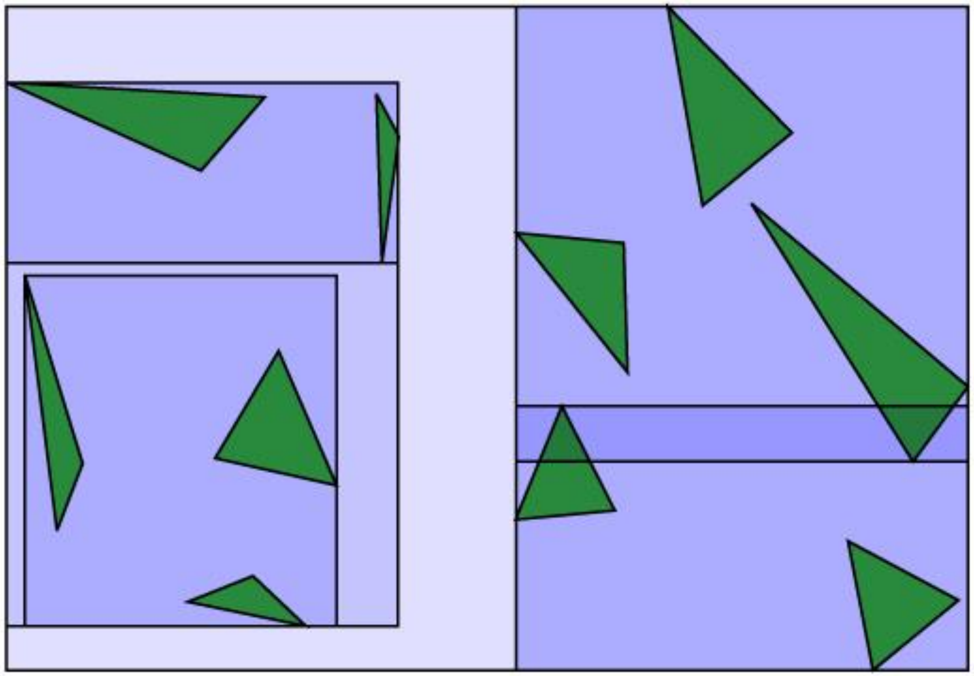
If it’s worth doing, it’s worth doing hierarchically!

- Bvols around objects may help
- Bvols around groups of objects will help
- Bvols around parts of complex objects will help
- Leads to the idea of using bounding volumes all the way from the whole scene down to groups of a few objects
- Meshes are a special case
  - We don’t add individual triangles into the DAG
  - We probably want to do something hierarchically at the level of an individual triangle soup!

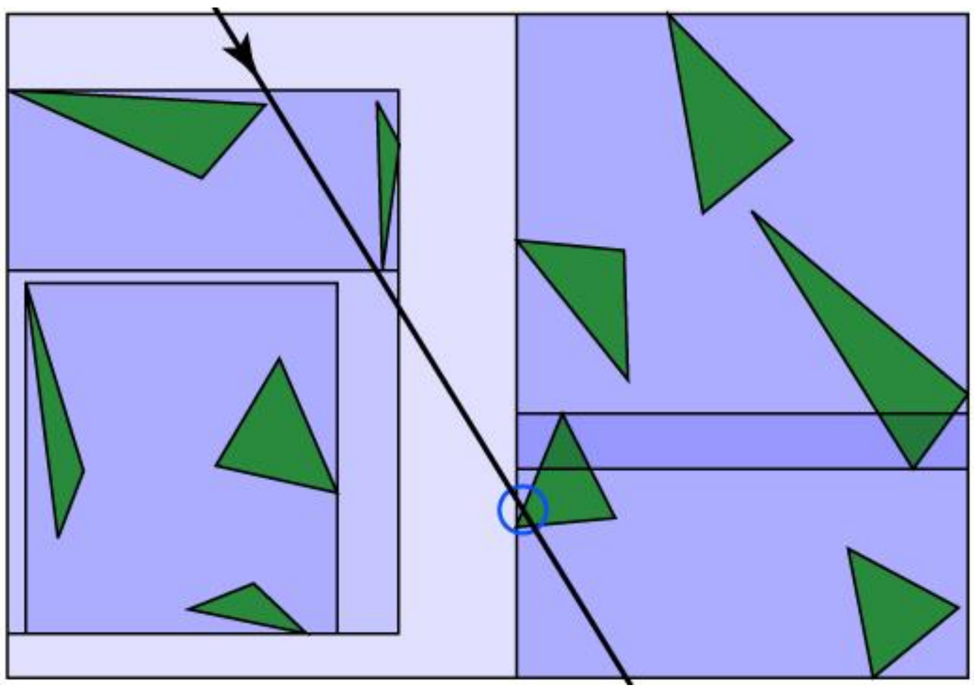
Implementing a bvol hierarchy

- A BoundedSurface can contain a list of Surfaces
- Some of those Surfaces might be more BoundedSurfaces
- Voilà! A bounding volume hierarchy
  - And it's all still transparent to the renderer

BVH construction example



BVH ray-tracing example



Based on slides by Steve Marschner

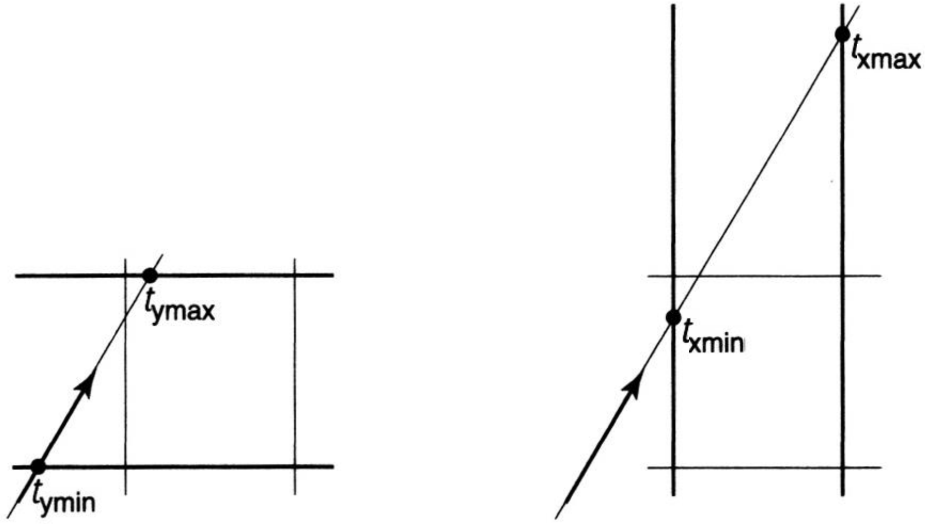
Choice of bounding volumes

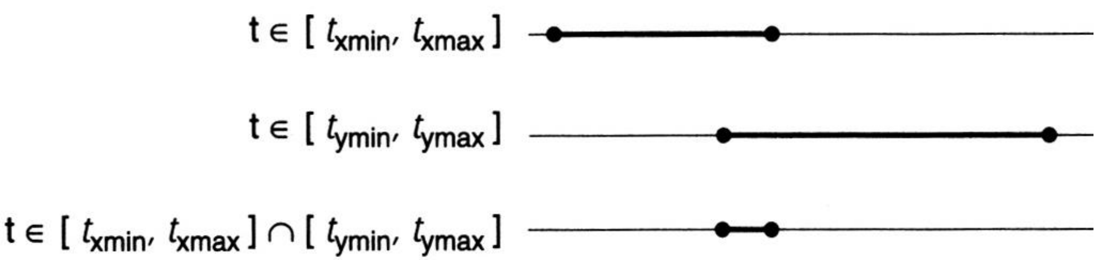
- Spheres
  - easy to intersect, not always so tight,
  - Tricky to compute tight bounds (google Welzl's smallest enclosing disk algorithm)
- Axis-aligned bounding boxes (AABBs) –
  - easy to intersect, often tighter (esp. for axis-aligned models)
- Oriented bounding boxes (OBBs) –
  - easy to intersect (but cost of transformation), tighter for arbitrary objects
- Computing the bvols
  - For primitives -- generally pretty easy
  - For groups -- not so easy for OBBs (to do well)
  - For transformed surfaces -- not so easy for spheres

Axis aligned bounding boxes

- Probably easiest to implement
- Computing for primitives
  - Cube: very straightforward!
  - Sphere, cylinder, etc.: pretty obvious
  - Groups or meshes: min/max of component parts
- AABBs for transformed surface
  - Easy to do conservatively: bbox of the 8 corners of the bbox of the untransformed surface
- How to intersect them
  - Treat them as an intersection of slabs (see Shirley)

Intersecting boxes





Building a hierarchy

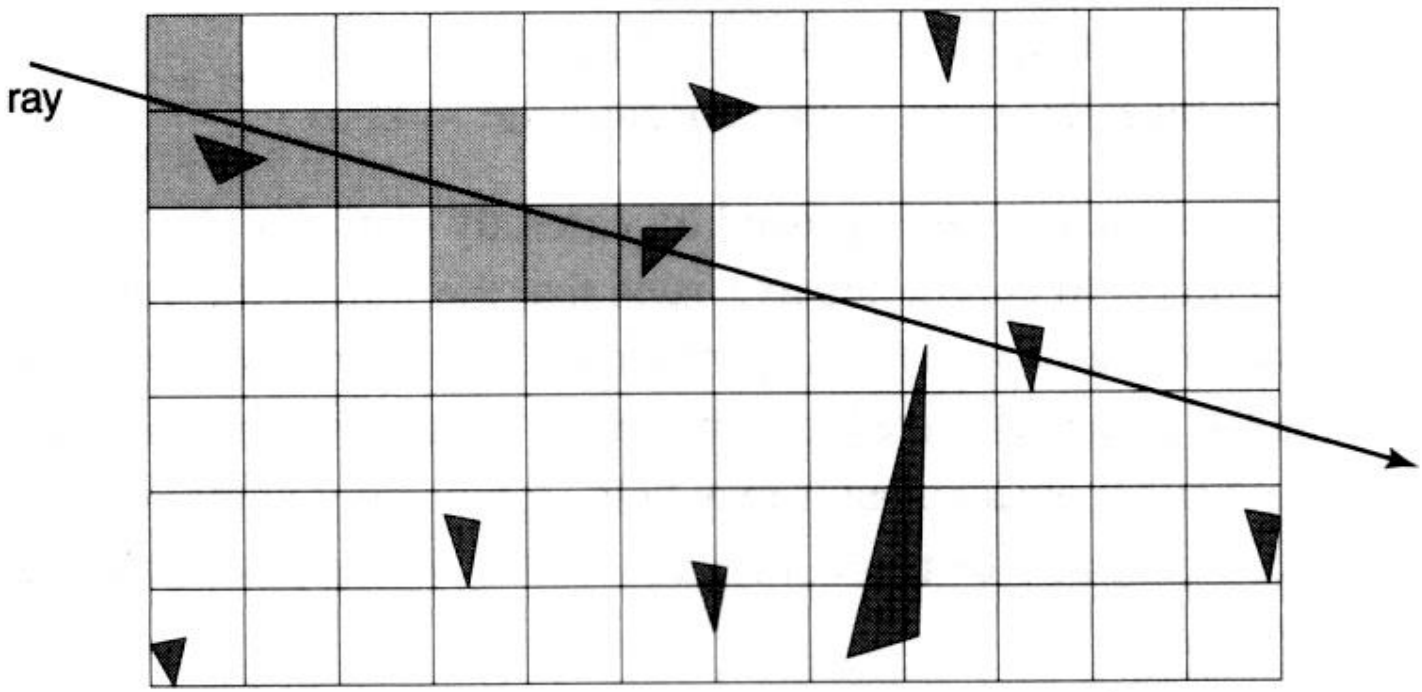
- Usually do it top-down
- Make bbox for whole scene, then split into (maybe 2) parts
  - Recurse on parts
  - Stop when there are just a few objects in your box

Building a hierarchy

- How to partition?
  - Ideal: clusters
  - Practical: partition along axis
    - Median partition
      - More expensive
      - More balanced tree
    - Center partition
      - Less expensive, simpler
      - Unbalanced tree, but that may actually be better

Regular space subdivision

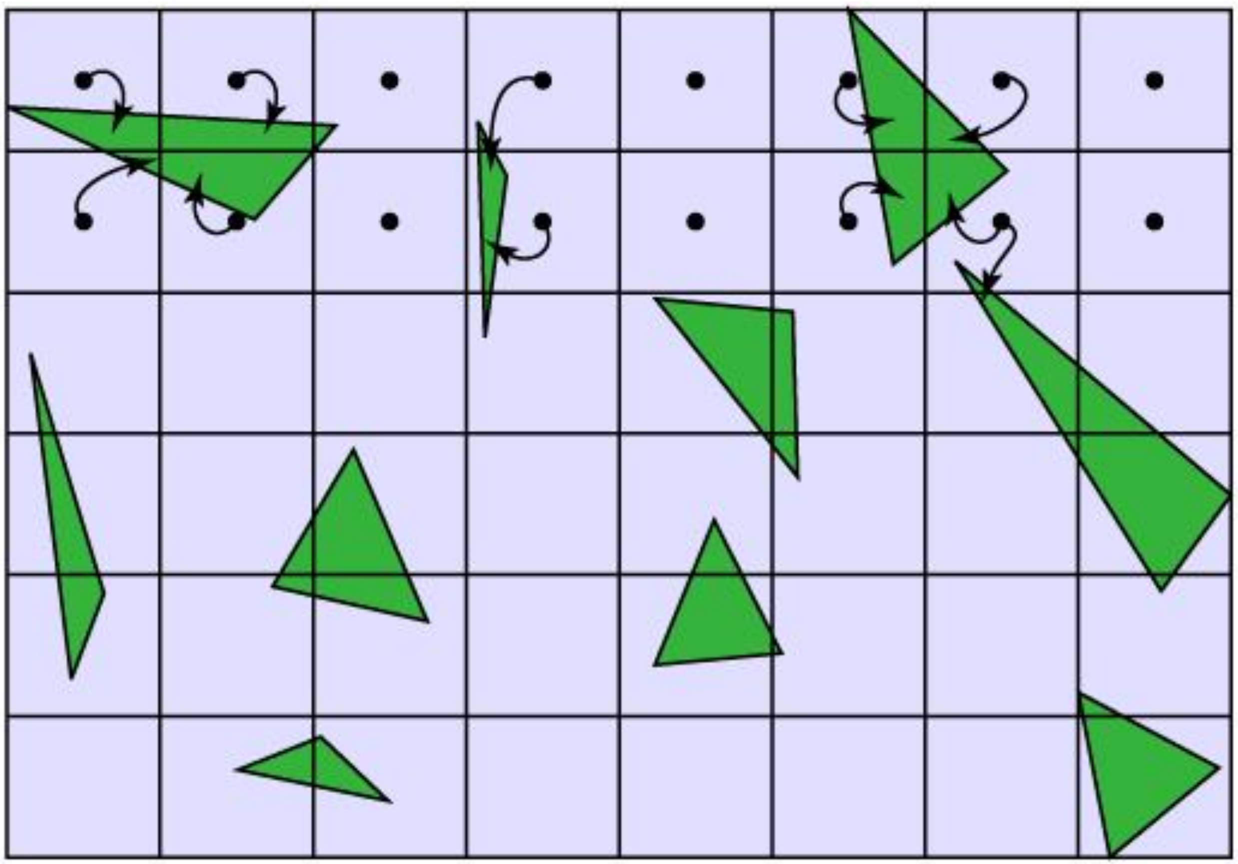
- An entirely different approach: uniform grid of cells



Based on slides by Steve Marschner

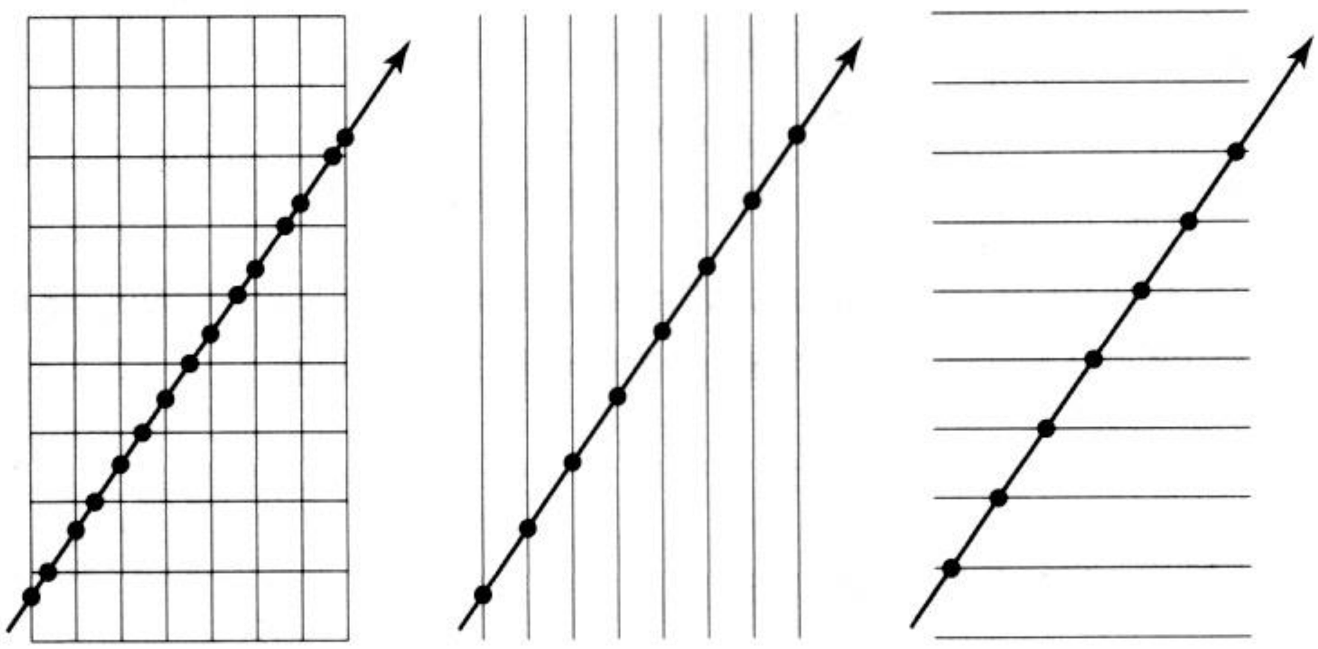
Regular grid example

- Grid divides space, not objects



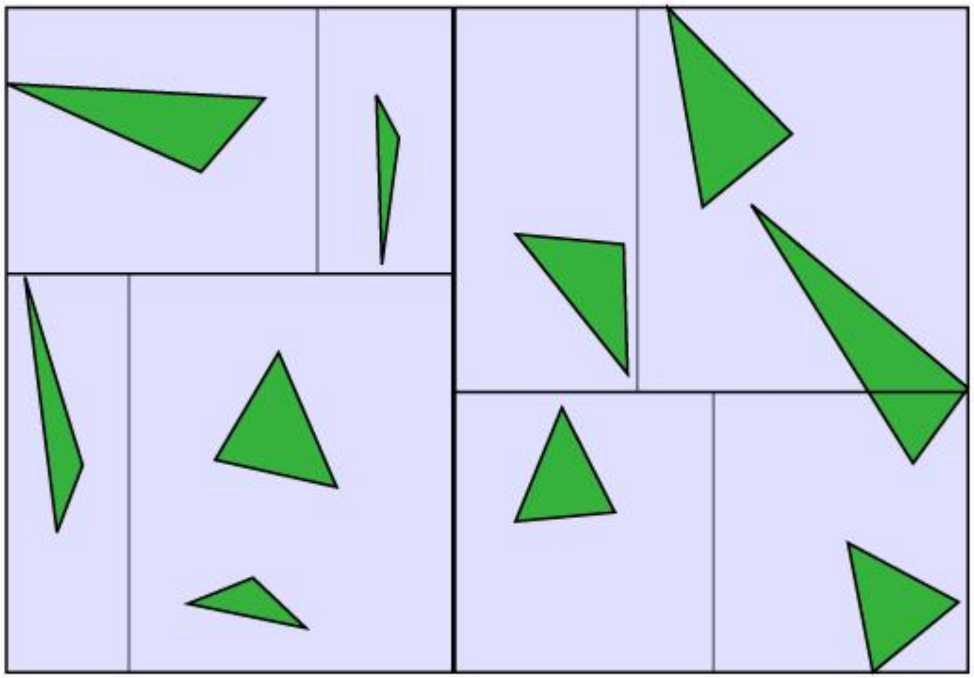
Based on slides by Steve Marschner

Traversing a regular grid



Non-regular space subdivision

- *k*-d Tree
  - subdivides space, like grid
  - adaptive, like BVH



Based on slides by Steve Marschner

Implementing acceleration structures

- Conceptually simple to build acceleration structure into scene structure
- Better engineering decision to separate them

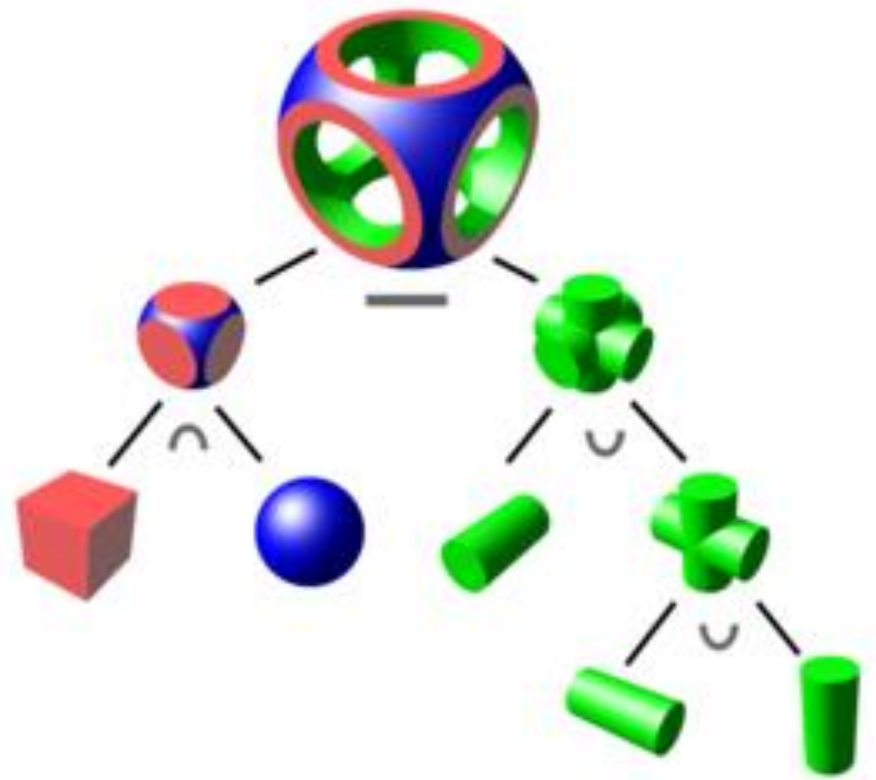
Other ray tracing features

- Fresnel reflection and refraction
- Area light sources / soft shadows
- Super sampling for anti aliasing & jittering
- Depth of field blur / real lens modelling
- Motion blur
- Transparency and compositing
- Cartoon shaders
- Texture maps (colour, displacement, normal)
- Other implicit surfaces (metaballs)
- Subdivision surfaces / parametric surfaces
- Constructive solid geometry
- Perlin Noise

Based on slides by Steve Marschner

Constructive Solid Geometry

- Exactly like intersecting a cube using 3 slabs
- Assemble binary set operations in a tree (works nicely with a dag scene graph)
- Ray intersection class must maintain a list of ***all*** intersections instead of closest intersection.



Meta Balls

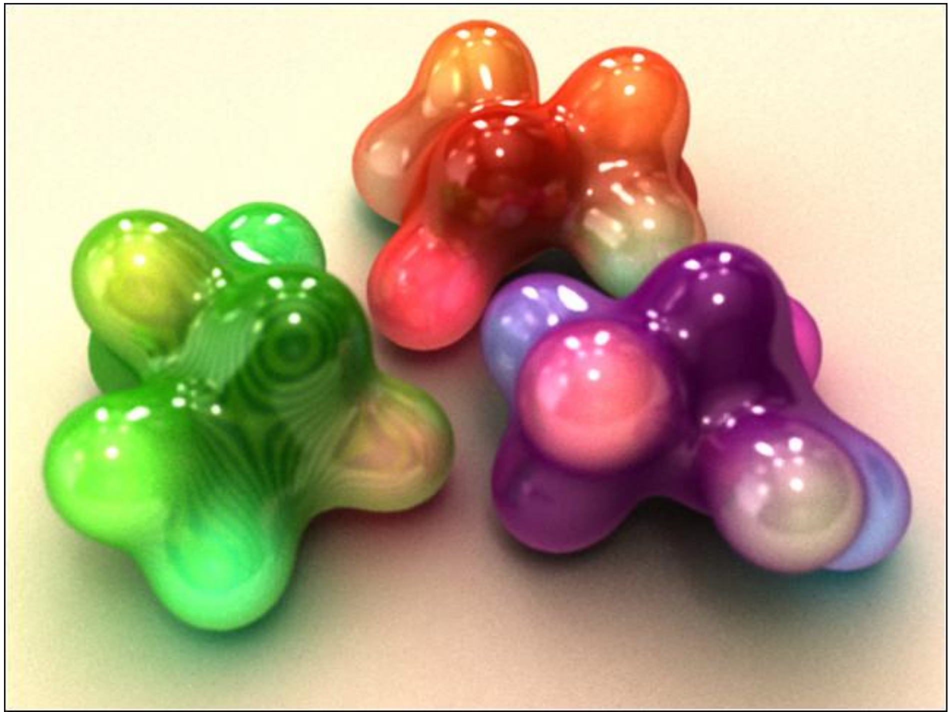
- Implicit function defines surface

$$\sum_{i=0}^{n} \text{metaball}_i(x, y, z) \leq \text{threshold}$$

$$f(x, y, z) = 1/((x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2)$$

  – Other better choices for functions too!

- ***How do you find the intersection?***
- ***What will be the normal?***



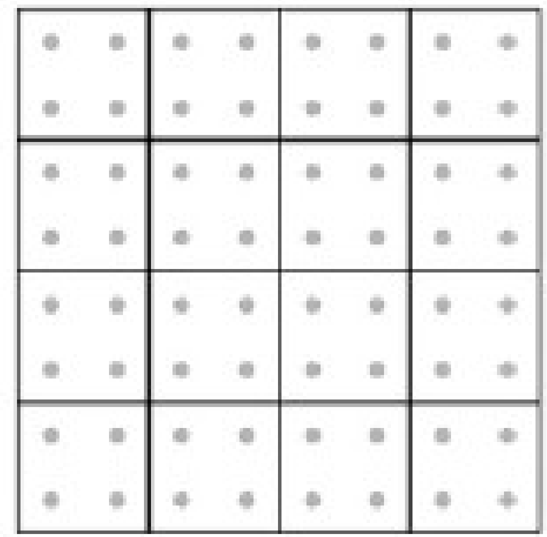
Perlin Noise

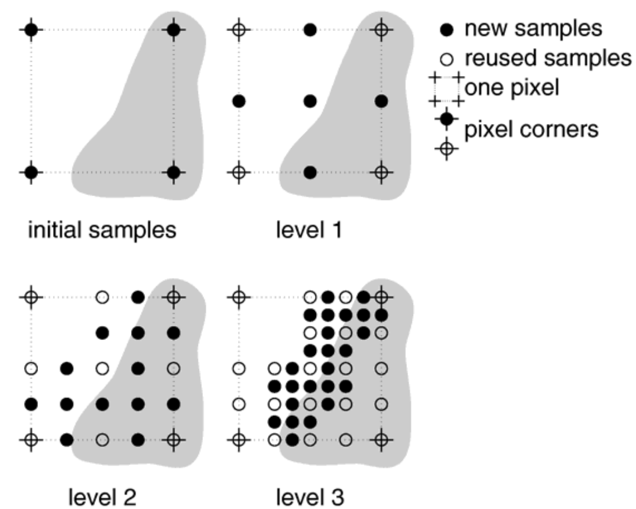
- Useful for clouds, marble, and other random varying effects
- Code is very small… noise is easy to compute!

Based on slides by Steve Marschner

Super Sampling and Jittering Samples

- Cast more rays through sub pixel positions.
- Watch out for aliasing!
  - Jittering samples will replace aliasing with noise, which is preferable
  - Noise added to sub-pixel position should not be too large!
- Super sampling can be done adaptively based on scene complexity at the pixel

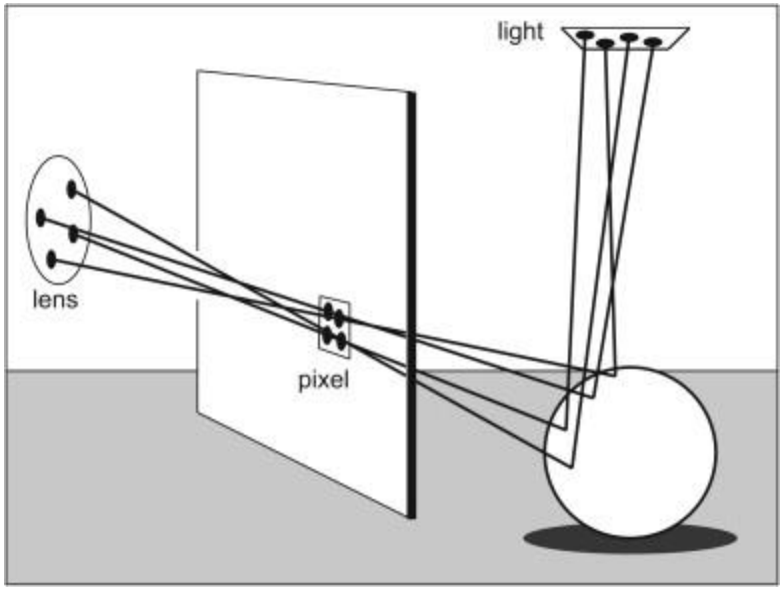


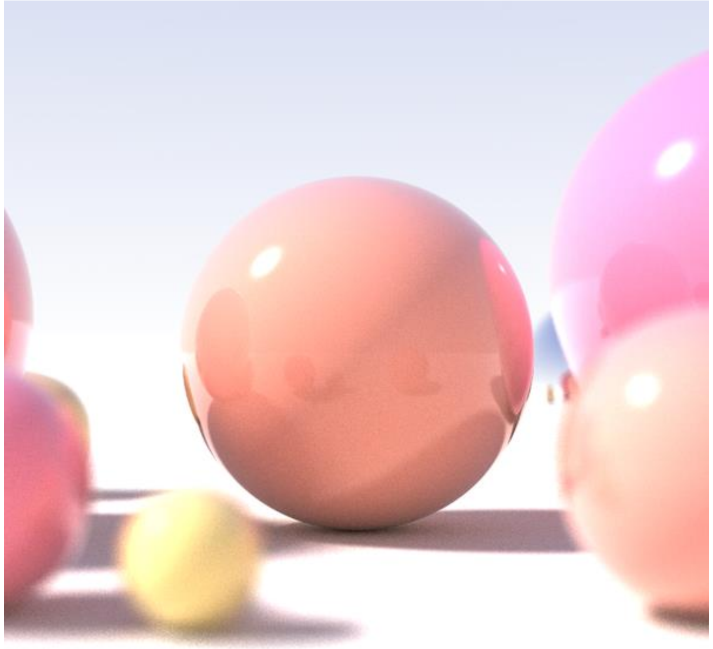


Based on slides by Steve Marschner

Depth of Field, Area Lights

- Cast more rays!
  - Sample different random positions on lens aperture by changing the eye position
  - For an area light source, do shadow ray test and lighting computation for a different random positions on the area light source.



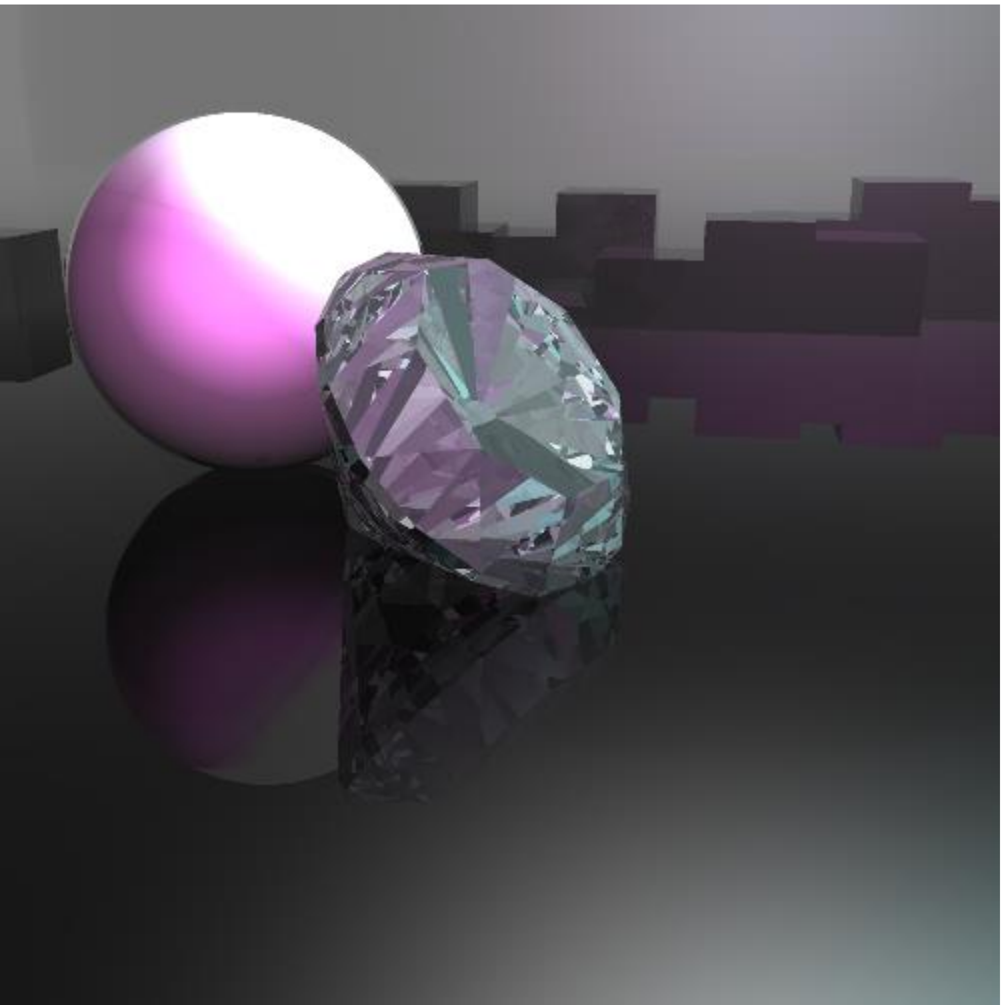


Motion Blur

- Knowing object trajectories over a time interval, cast rays at different times and average.
- Can also just average a sequence of images, but this would produce temporal aliasing!
- Can also be smart about not casting few rays where nothing is in motion.

Based on slides by Steve Marschner

Reflection and Refraction



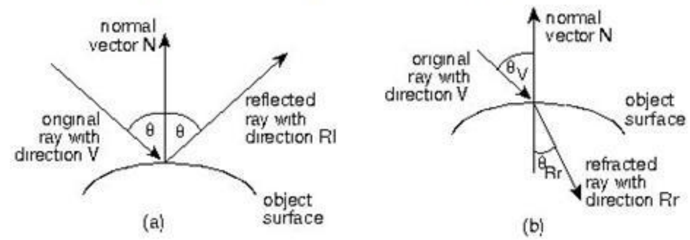
$$n1 = \text{index of refraction of original medium}$$

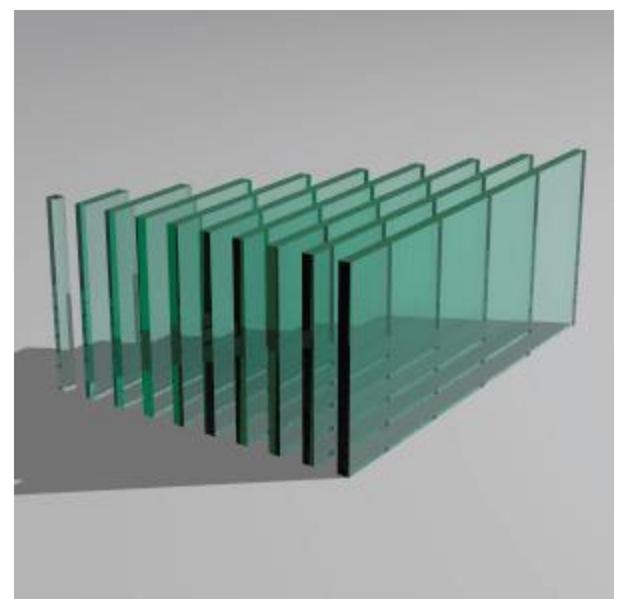
$$n2 = \text{index of refraction of new medium}$$

$$n = \frac{n1}{n2}$$

$$c2 = \sqrt[2]{1 - n^2 \times (1 - c1^2)}$$

$$Rr = (n \times V) + (n \times c1 - c2) \times SN$$

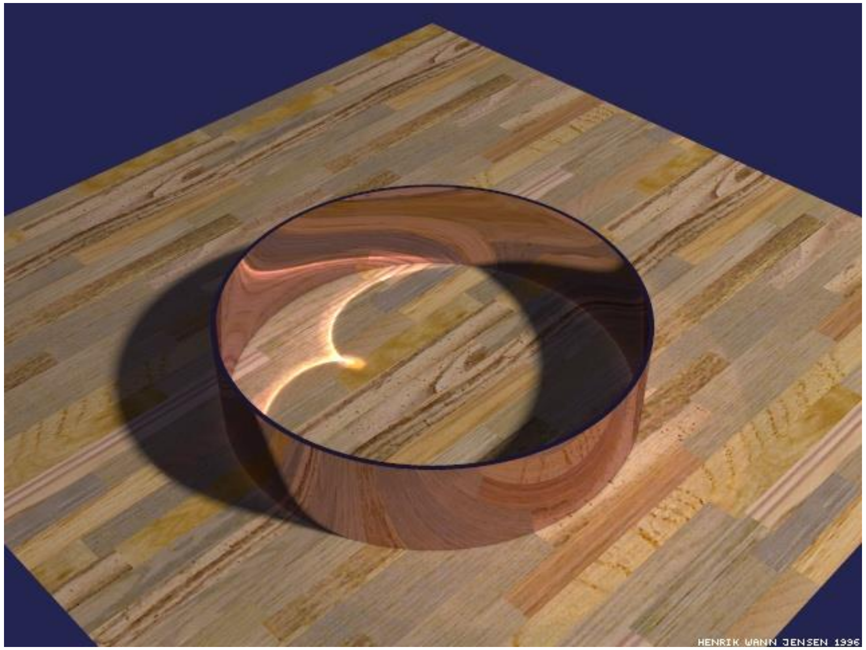


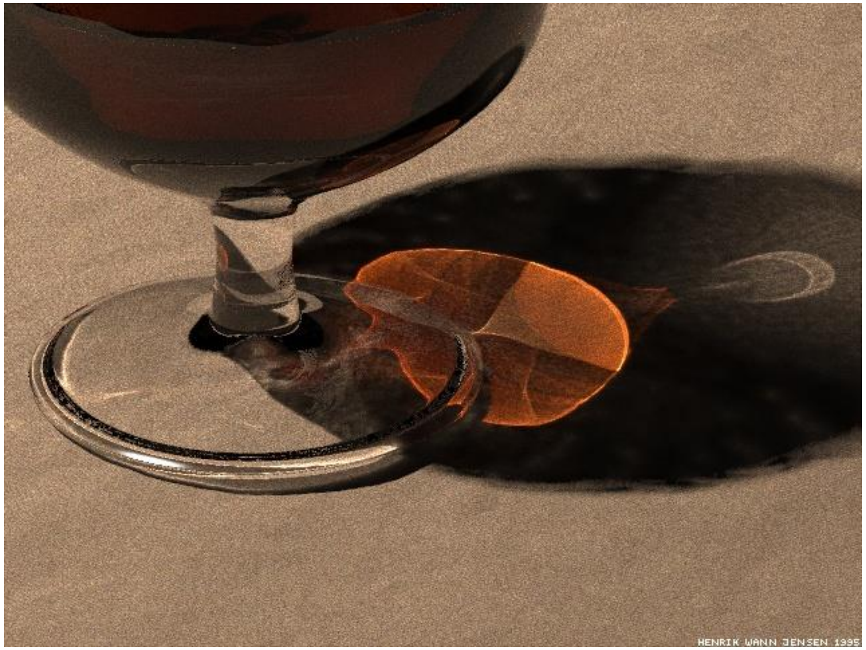


Based on slides by Steve Marschner

Caustics

- Reflection and refraction scatters light too





Based on slides by Steve Marschner

Anisotropic BRDFs

- Can use a varying reflectance function for computing the amount of reflected light.

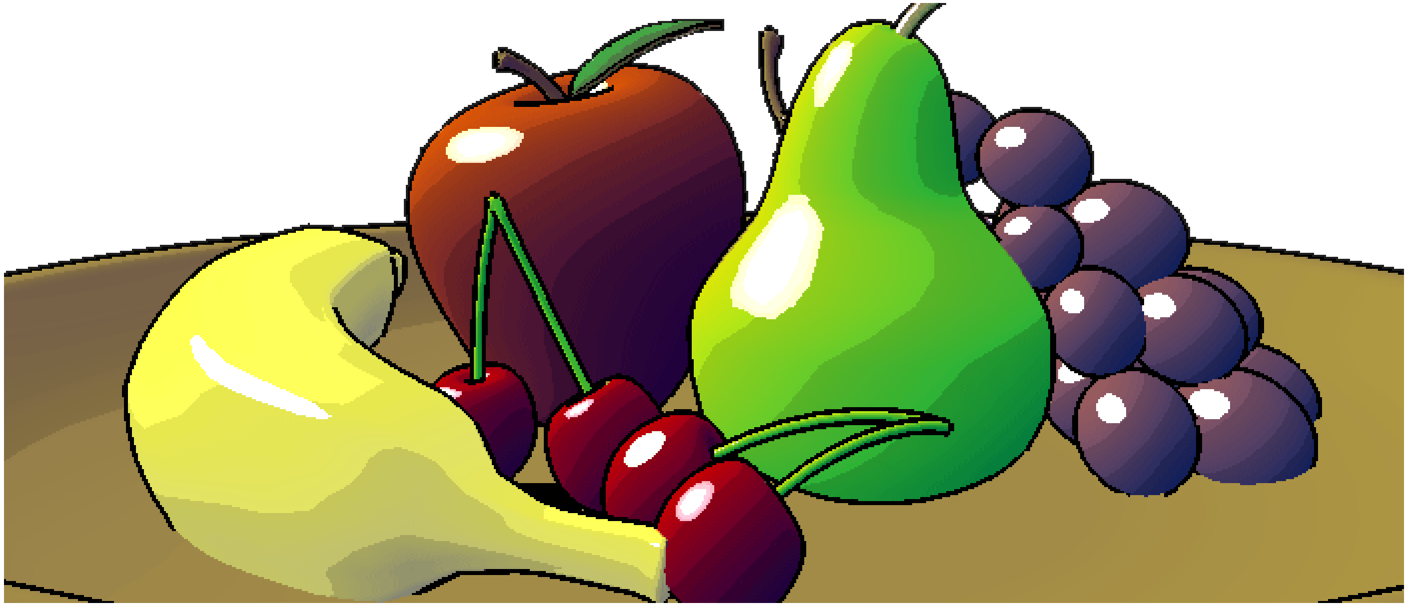
Based on slides by Steve Marschner

Transparency and Compositing

- Render an image and then combine it with a real photograph.
- Difficult to get lighting correct
  - Typically take a “light probe”, i.e., a photo of a shiny sphere
  - Use sphere image like an area light, and try to address the expensive sampling problem through importance sampling

Cartoon Shaders

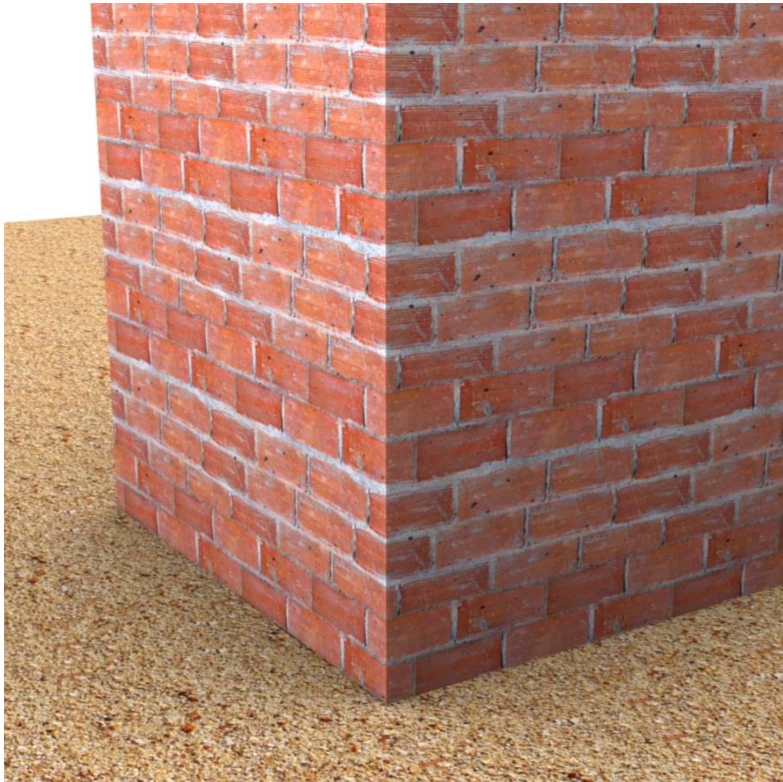
- Change lighting computations so that they produce a smaller number of colours.
- Add black edges at boundaries between objects.



Based on slides by Steve Marschner

Texture Maps

- Can download texture images from a variety of repositories.
- Easy to texture certain primitives (cubes, spheres, planes)
- Meshes often have texture coordinates attached
- ***Sampling is an issue!!***
  - Resolution may be too high or too low giving a poor quality result



Review and More Information

- Textbook
  - 12.3 Spatial Data Structures
  - 13.1 Transparency and Refraction
  - 13.2 Instancing
  - 13.3 Constructive Solid Geometry
  - 13.4 Distributed Ray Tracing
    - Antialiasing, soft shadows / area lights, depth of field, glossy reflection (motion blur not in text but similar)