Ray Casting

MIT EECS 6.837

Frédo Durand and Seth Teller

Some slides courtesy of Leonard McMillan

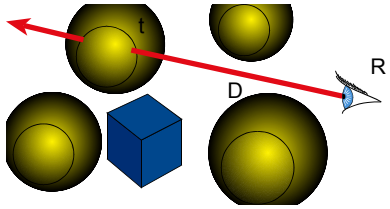
Administrative

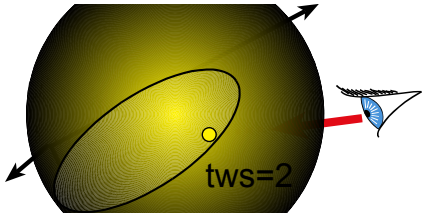
- Assignment 4
  - Due Friday 11 at 5pm
  - Polygon rasterization
  - Phong highlight

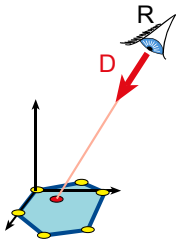
Questions?

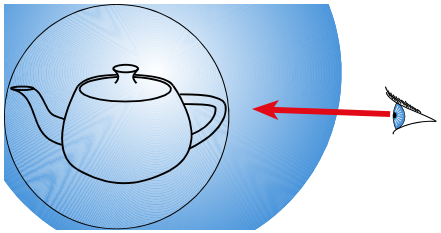
Overview

- Ray casting
- Transformations
- Ray-Primitive intersection
- Advanced ray casting







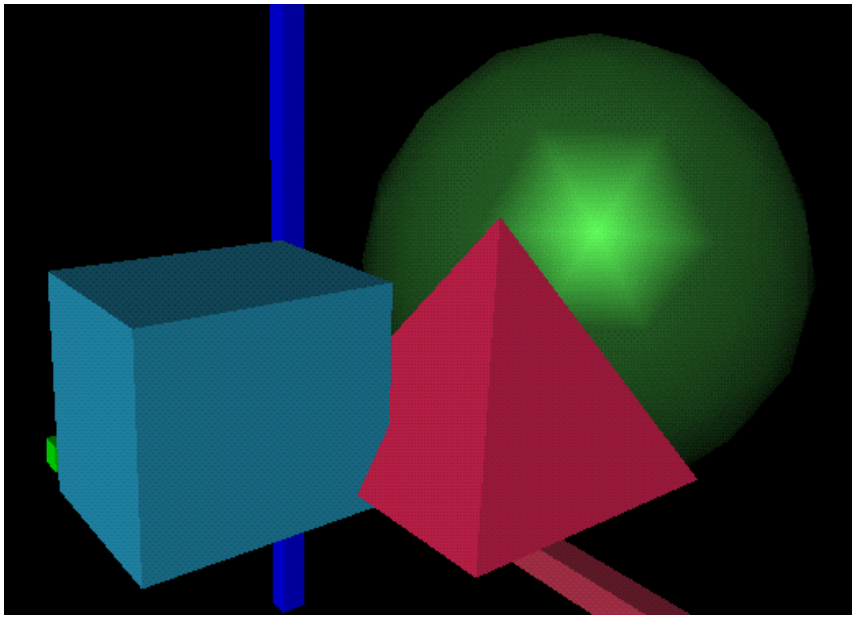


Scan Conversion and Visibility

- Hidden surfaces eliminated at each pixel
  - scanline algorithms
  - z buffering
  - Shading computed many times at each pixel
  - Each pixel stores only constant state

Limitations of Scan Conversion

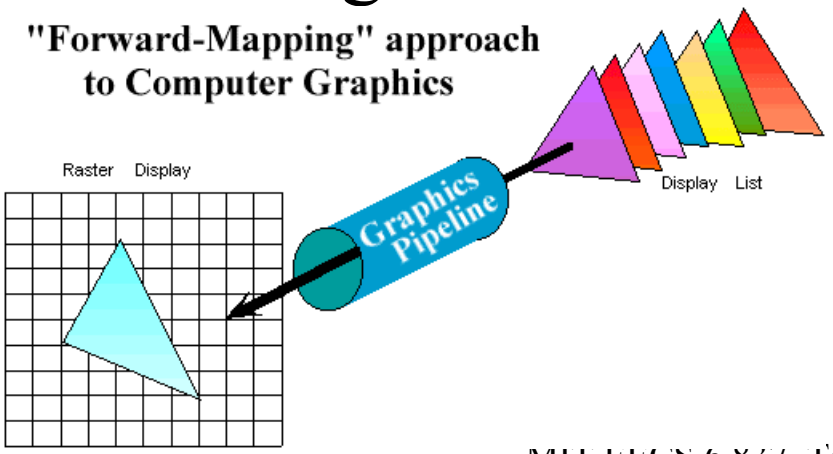
- Restricted to scan-convertable primitives
  - Object polygonalization
- Faceting, shading artifacts
- Effective resolution hardware dependent (aliasing)



- No handling of shadows, reflection, transparency
- What if there are more triangles than pixels?
- Problem of overdraw (high depth complexity)

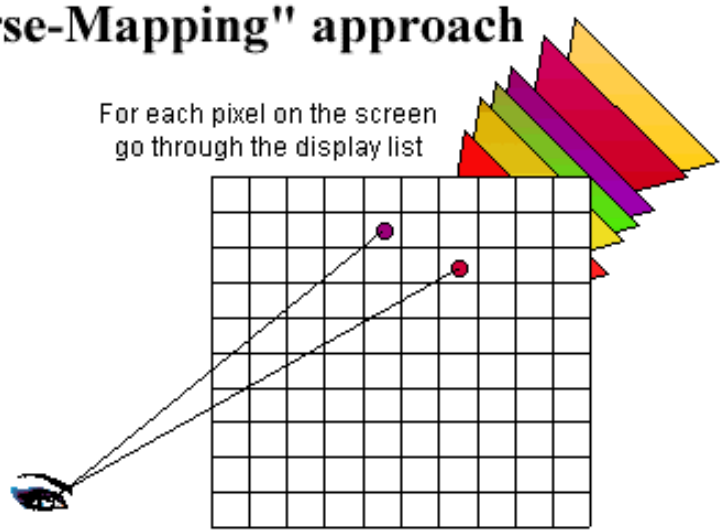
Graphics Pipeline Review

- Primitives are processed one at a time
- All analytic processing early on
- Sampling occurs late
- Minimal state required (immediate mode rendering)
- Processing is forward-mapping



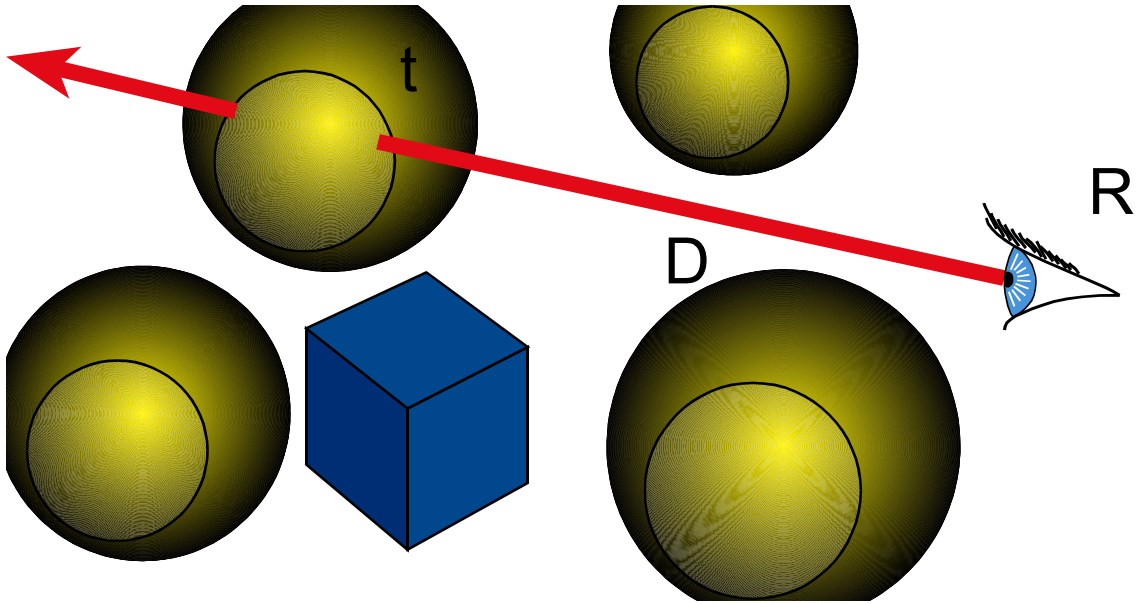
Ray Casting Outline

- For every pixel construct a ray from the eye
- For every object in the scene
  - Find closest intersection with the ray
  - Compute normal at point of intersection
- Compute color for pixel (e.g. Phong)



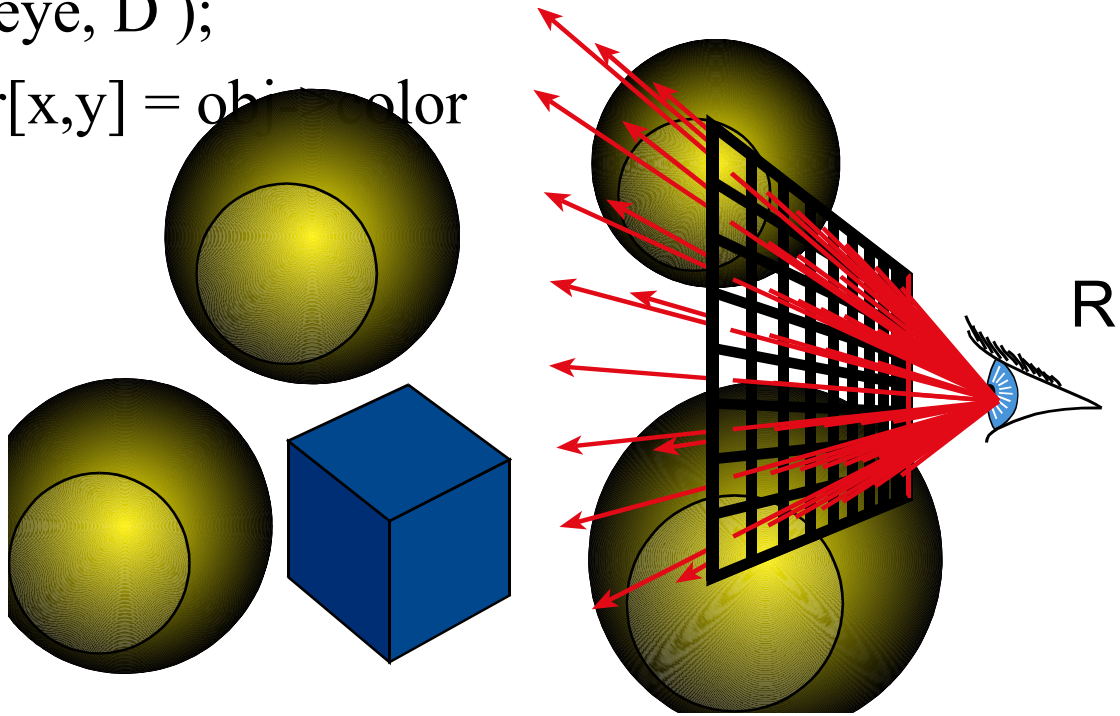
Ray Casting

```
Object Cast ( Point R, Ray D ) {
    find minimum t>0 such that R + t D hits object
    if ( object hit )
        return object
    else  return background object
    }
```



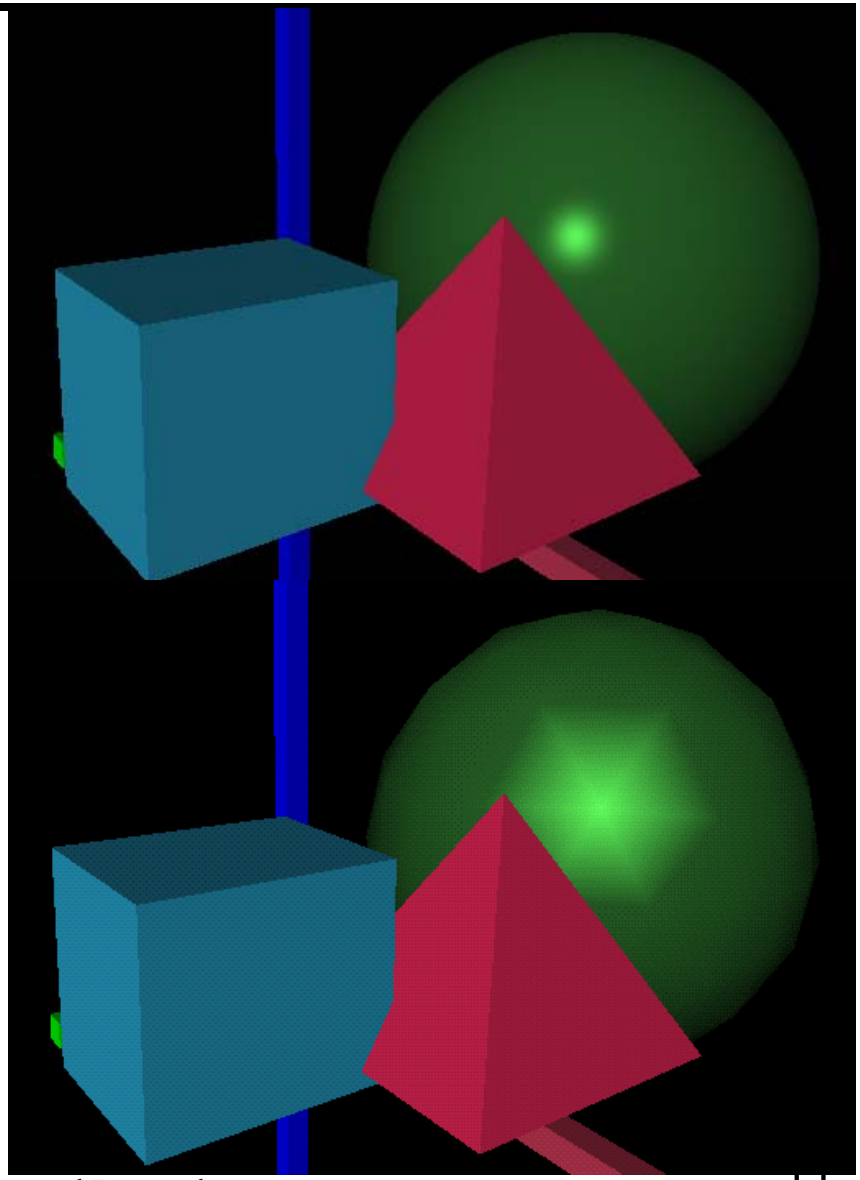
Ray Casting

```
FrameBuffer Render ( frustum, viewport ) {
  For each raster y
    For each pixel x
      D = ray from eye through pixel x, y
      obj = Cast ( eye, D );
      FrameBuffer[x,y] = obj.color
}
```



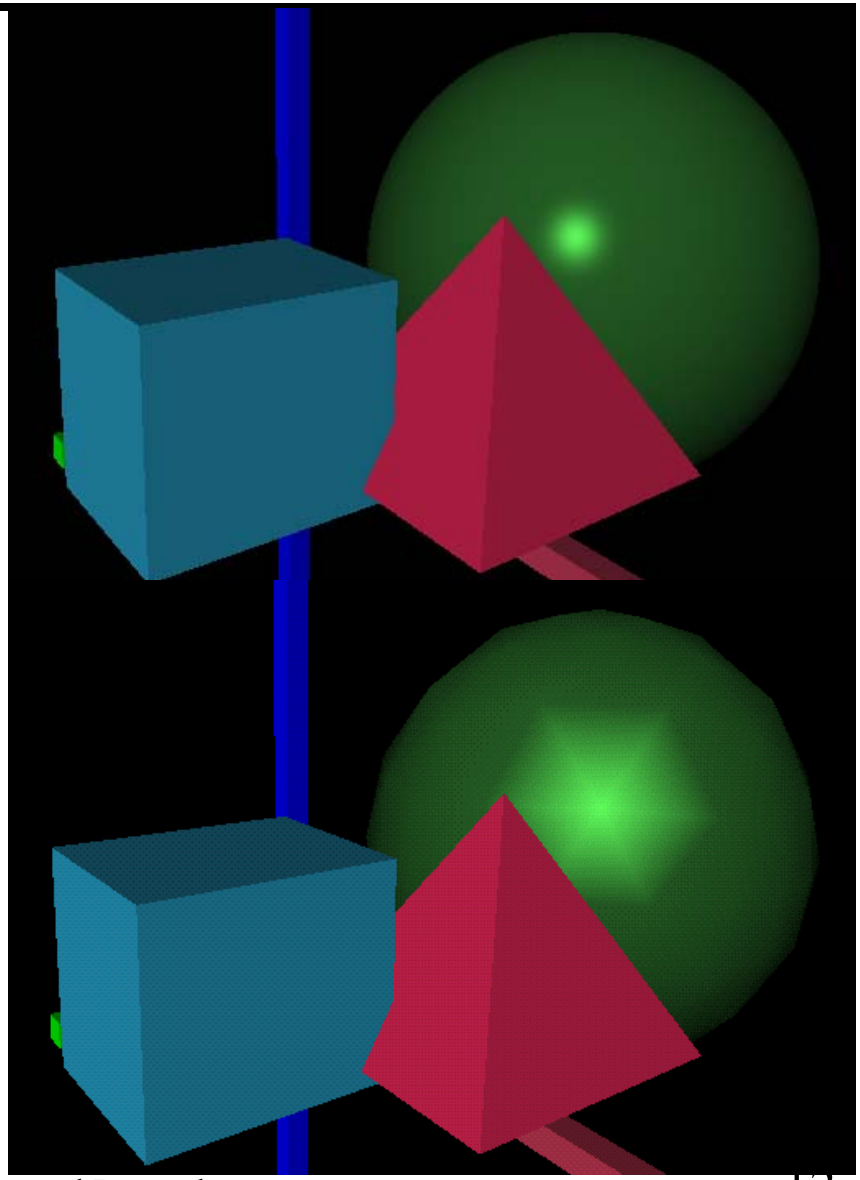
Ray Casting for Visibility

- Advantages?
  - Smooth variation of normal, silhouettes
  - Generality: can render anything with which a ray can be intersected !
  - Compactness of representation
  - Atomic operation



Ray Casting for Visibility

- Disadvantages?
  - Time complexity (N objects, R pixels)
  - ?
  - Wasted work at “background" pixels
  - Why aren't ray casters usually found in hardware?

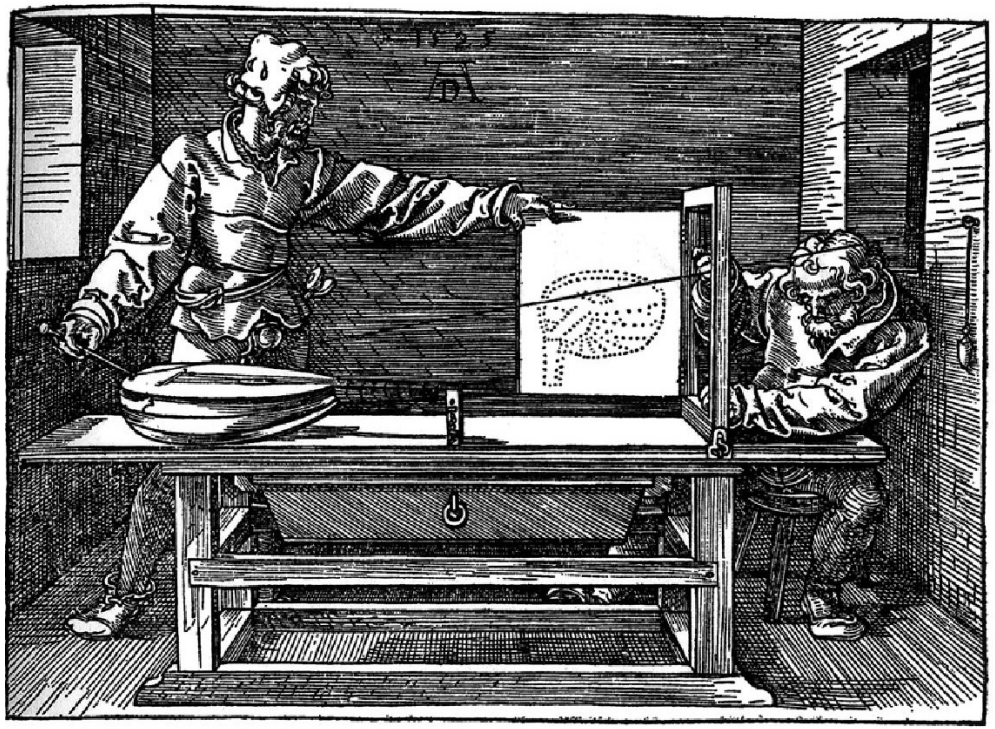


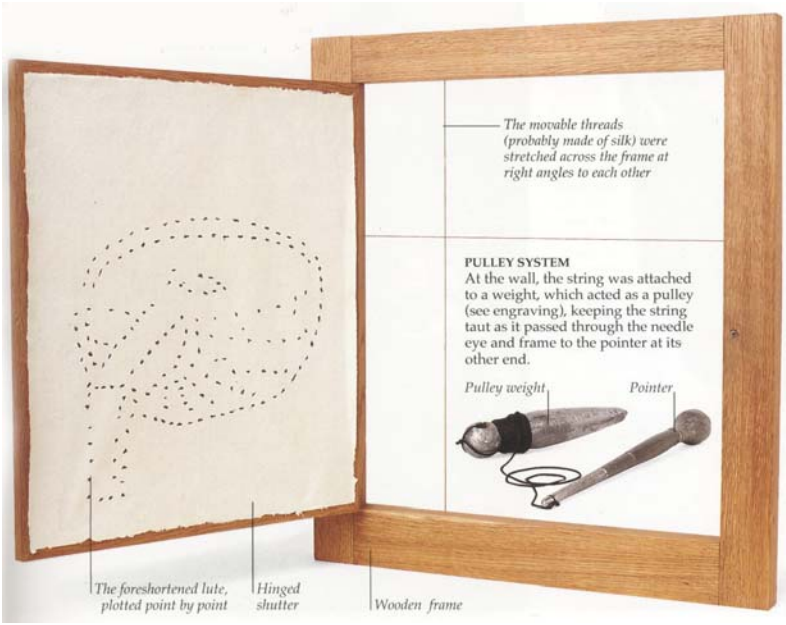
Ray Casting vs. Scan conversion

- Ray casting:
  - More and more used in production
    - Final Fantasy, The Cell, Fight Club, The City of Lost Children
  - Some hardware is appearing
  - Volume Rendering
- Scan conversion
  - Graphics card
  - Pixar Renderman
    - Toy Story, Monsters, Jurassic Park, Most of Final Fantasy

Durer's Ray casting machine

- Albrecht Durer, 16th century



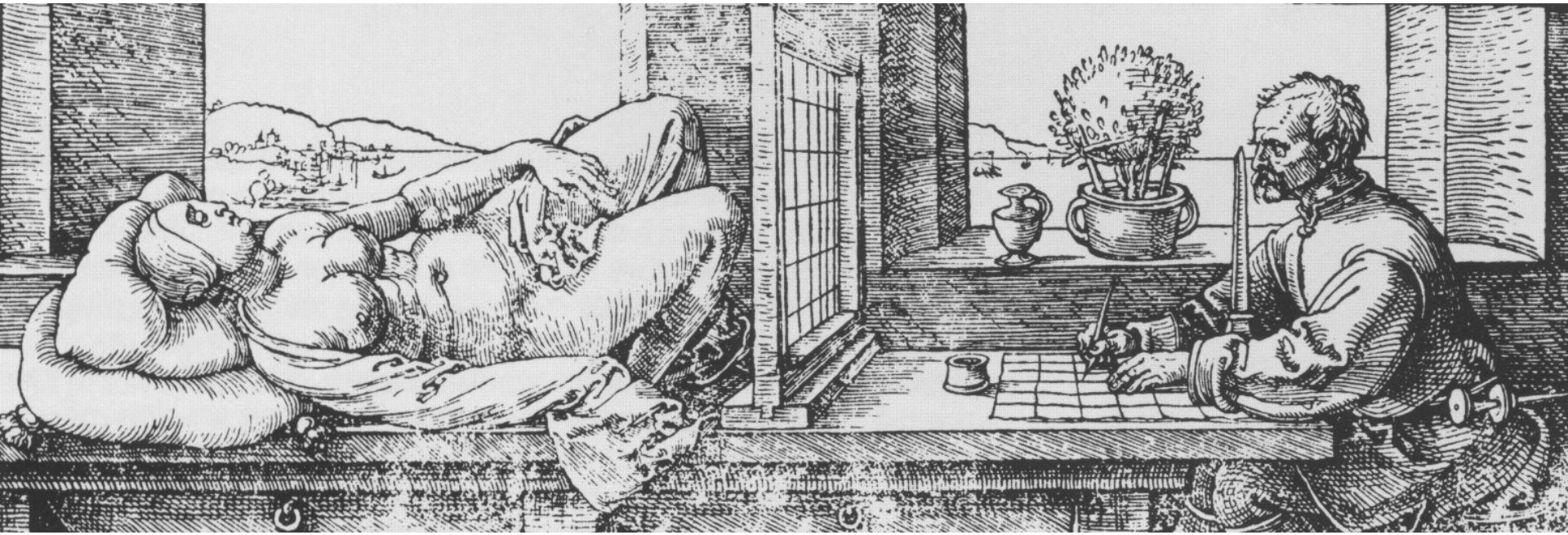


Durer's Ray casting machine

- Albrecht Durer, 16th century

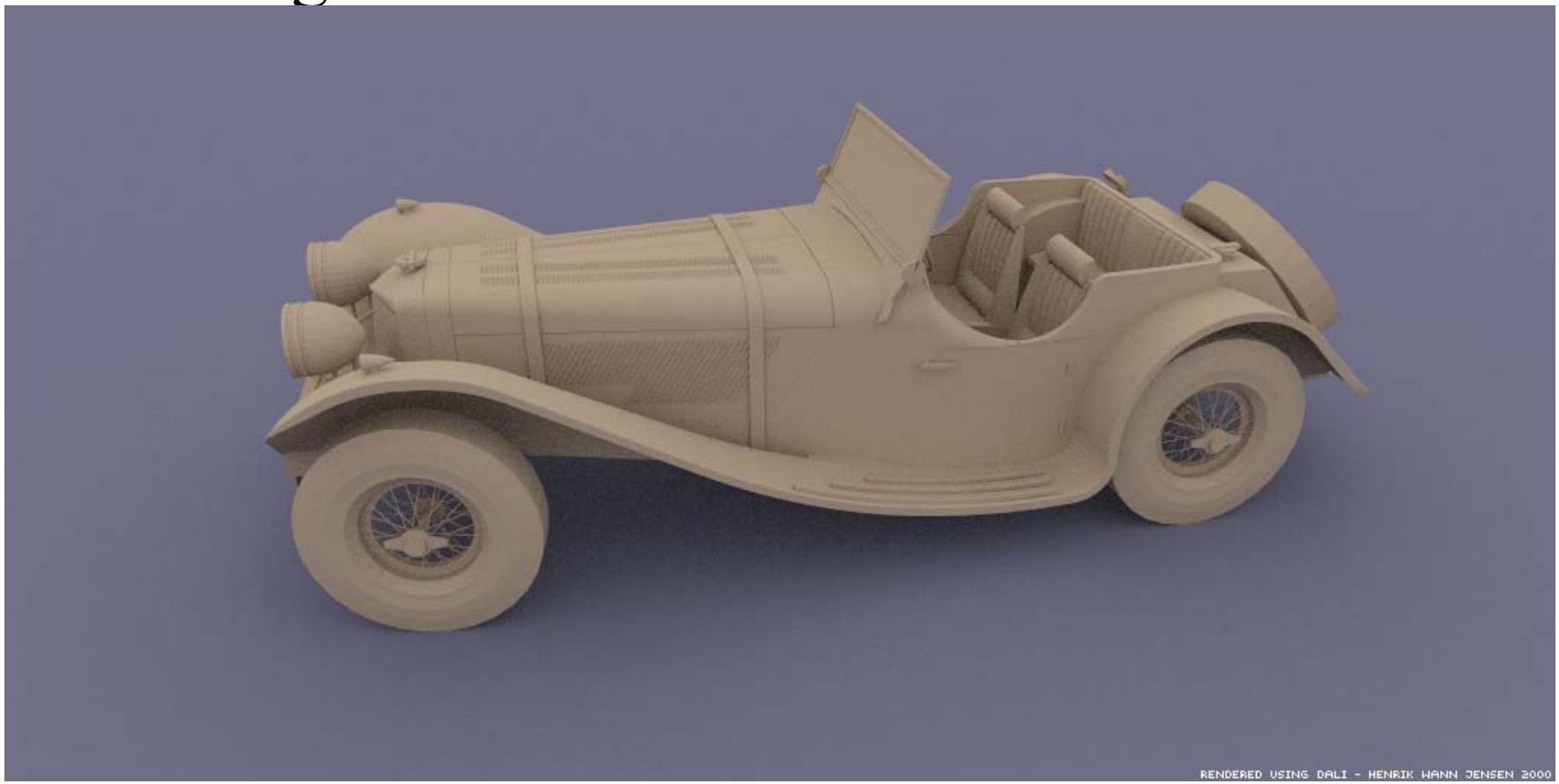
Durer's Ray casting machine

- Albrecht Durer, 16th century



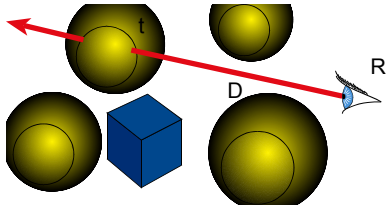
Questions?

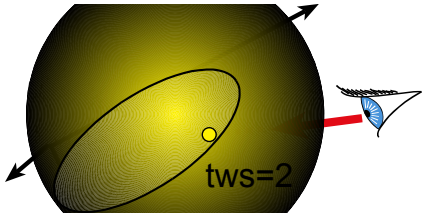
- Image by Henrik Wann Jensen using Ray Casting

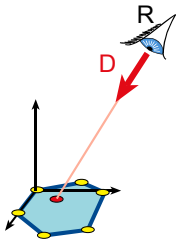


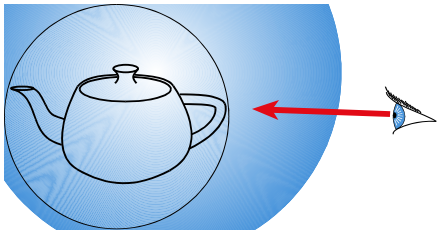
Overview

- Ray casting
- Transformations
- Ray-Primitive intersection
- Advanced ray casting



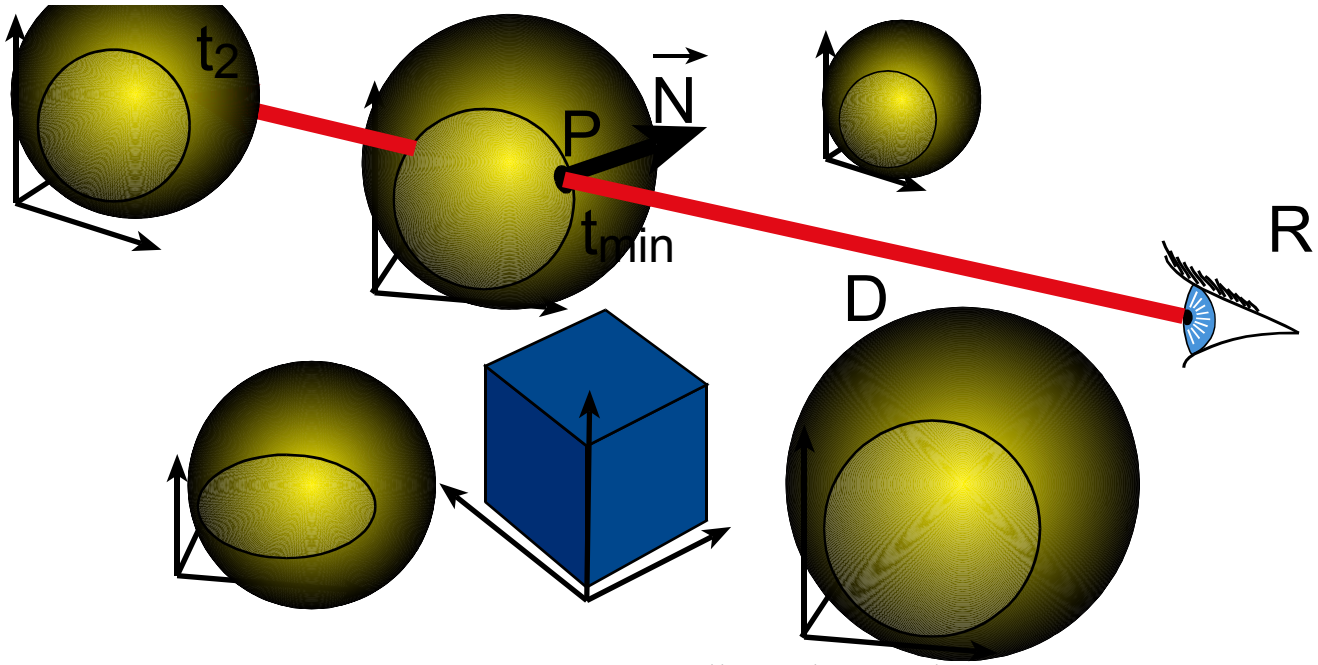






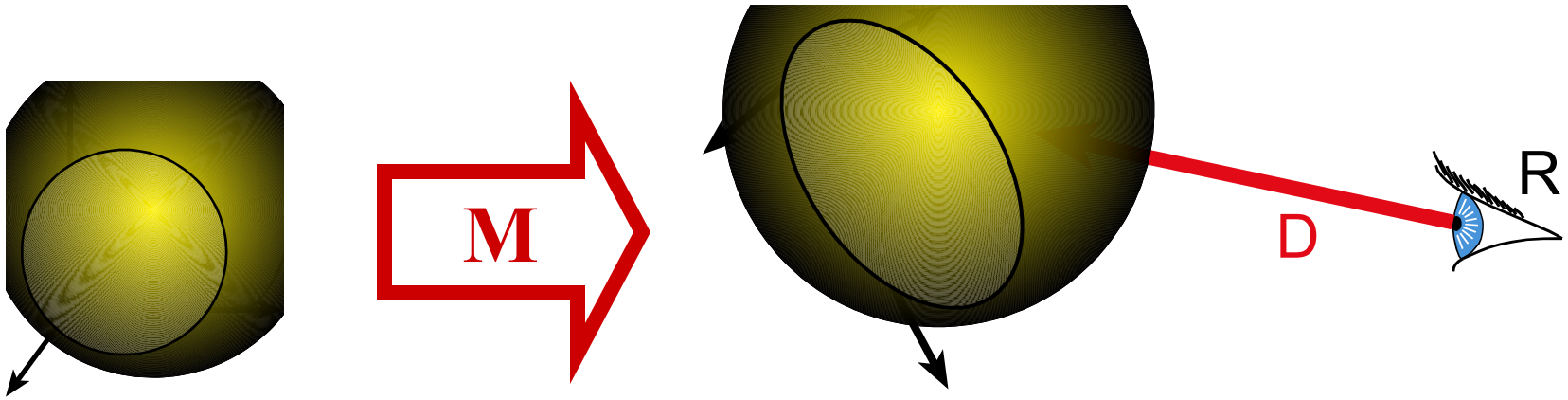
Ray-Scene Intersection

- Given a ray, & a scene of transformed primitives
  - Determine:
    - Closest ray-primitive intersection P, if any
    - Primitive's surface normal N at point P
- Compute *t* for each intersection, and retain tmin



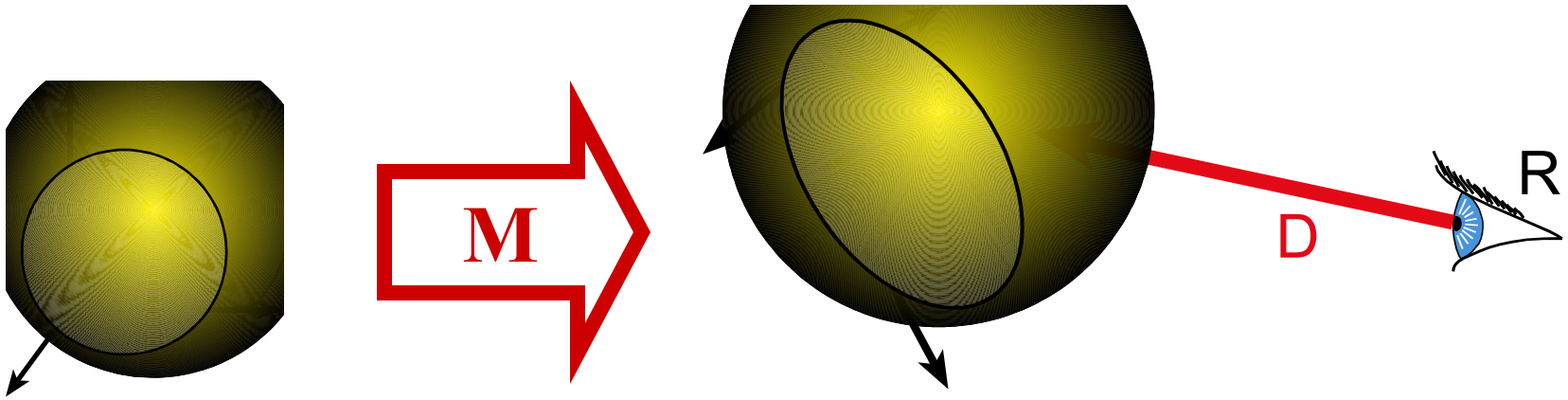
Ray-Primitive Intersection

- First concentrate on individual primitives
  - Ray is typically generated in world space
- Primitives defined in object space, then transformed
- What are our options?



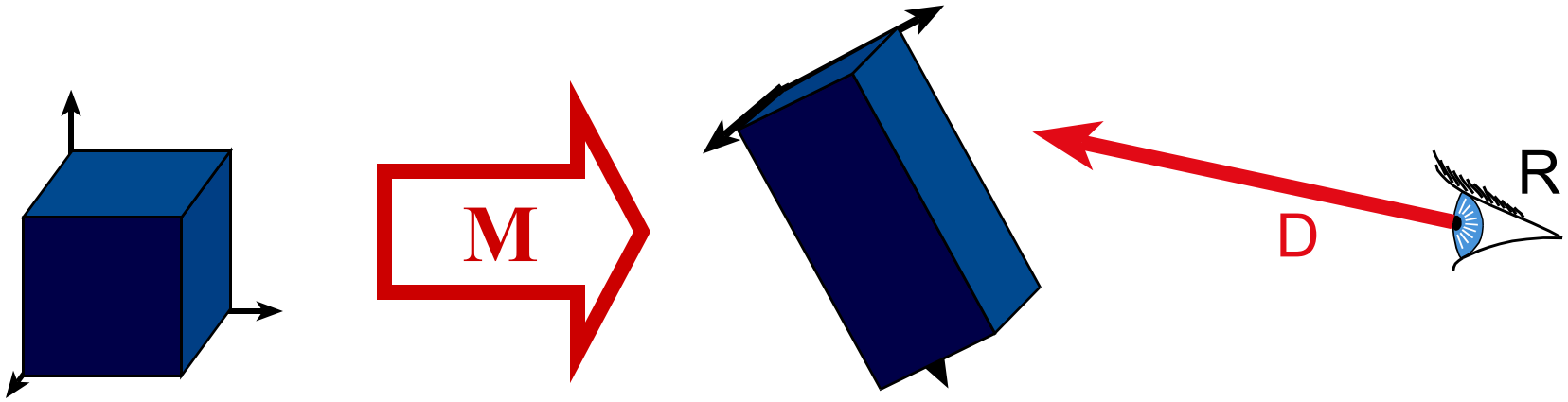
Opt 1: Intersection in World Space

- Transform primitive into world space
  - Reexpress it there, then perform intersection



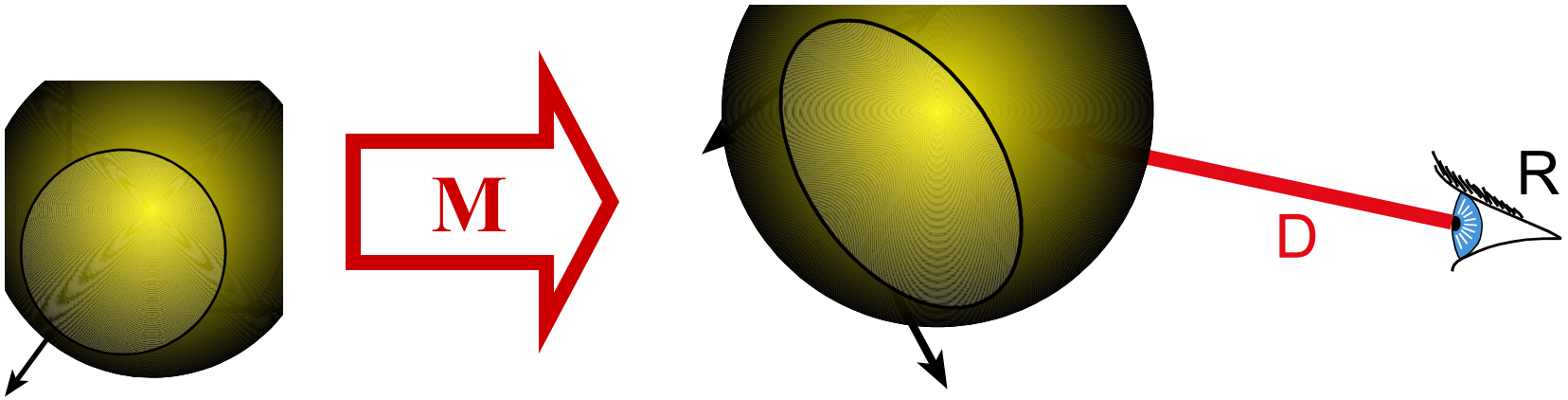
Opt 1: Intersection in World Space

- Option 1: transform primitive into world space
  - Reexpress it there, then perform intersection
- Polygon: vertices p as Mp, plane H as HM^{-1}
  - Polyhedra: collections of polygons



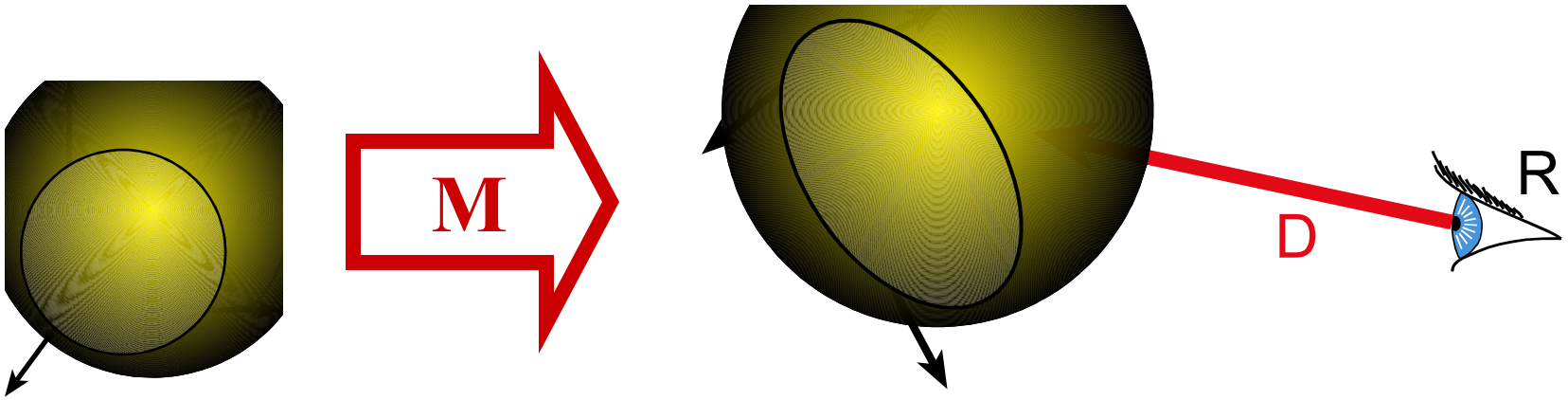
Opt 1: Intersection in World Space

- Option 1: transform primitive into world space
- Polygon: vertices as Mp, plane as HM-1
- Quadrics Q: transform as MQM^{-1}
  - But: quadric class is not invariant !
- Torii, spline surfaces, other primitives: harder.



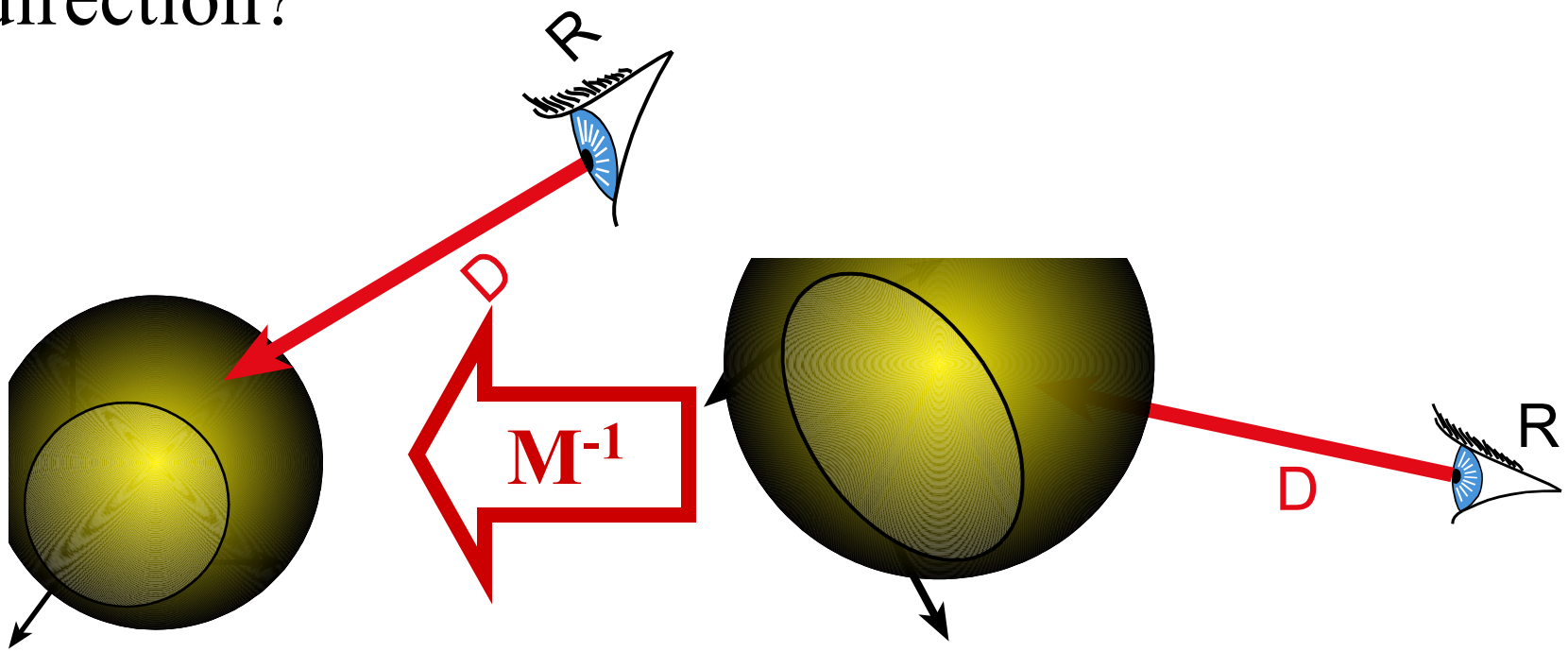
Opt 1: Intersection in World Space

- Option 1: transform primitive into world space
- Requires one transformation rule for each class of object
- Any better idea?



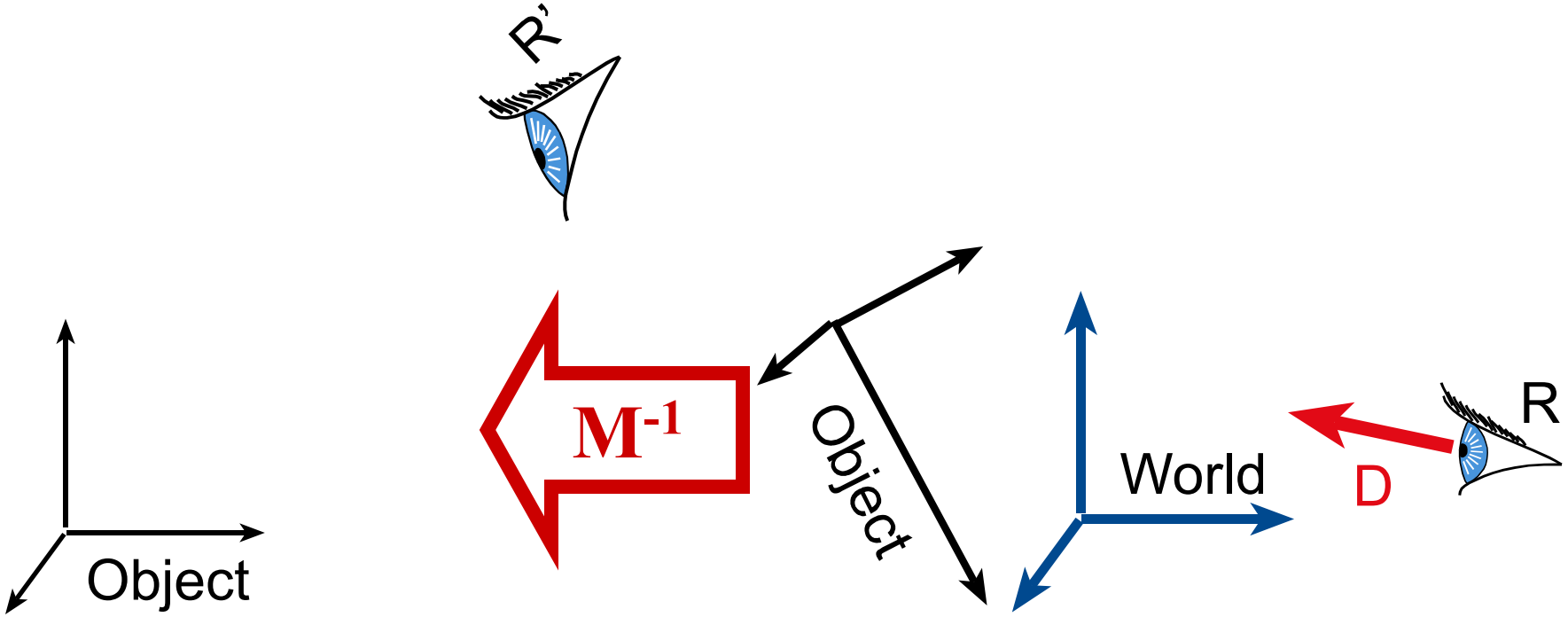
Opt 2: Intersection in Object Space

- Transform ray into object space!
  - Intersections straightforward, even trivial there
- How do we transform the ray's origin and direction?



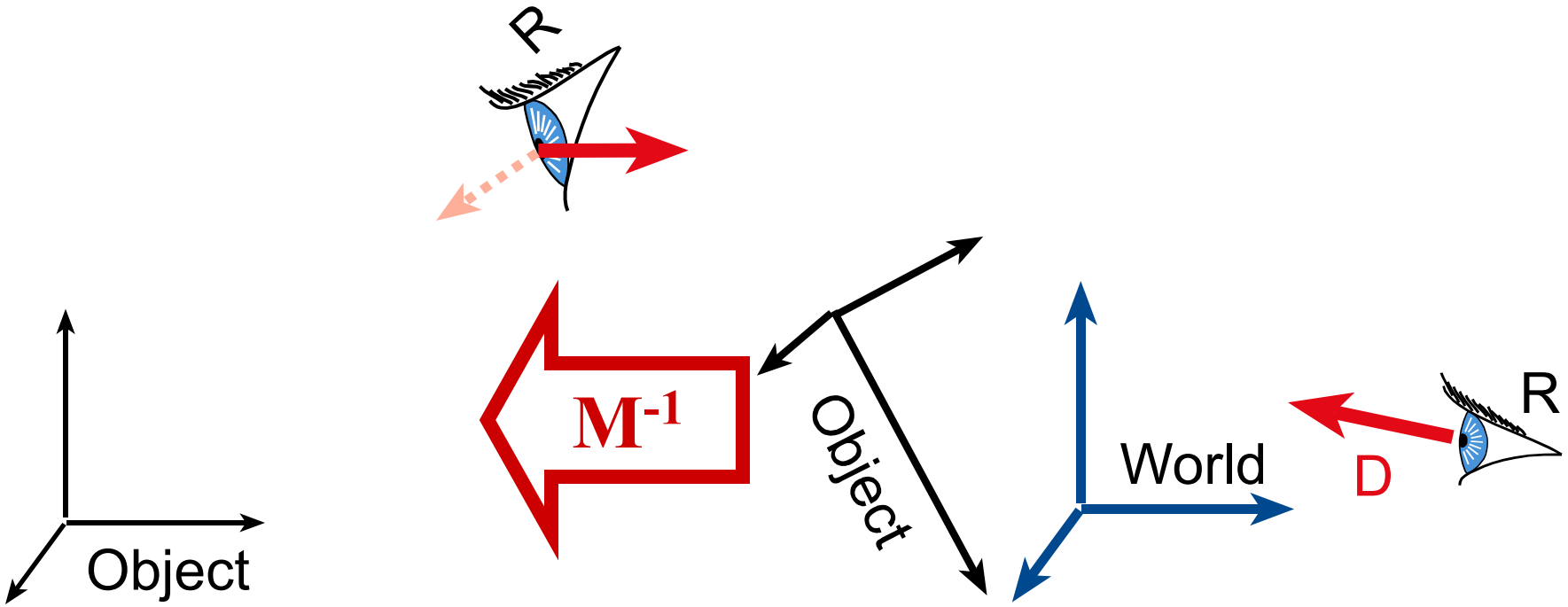
Opt 2: Intersection in Object Space

- Transform ray origin into object space
  - $R' = M^{-1}R$
- How do we transform the ray's direction?



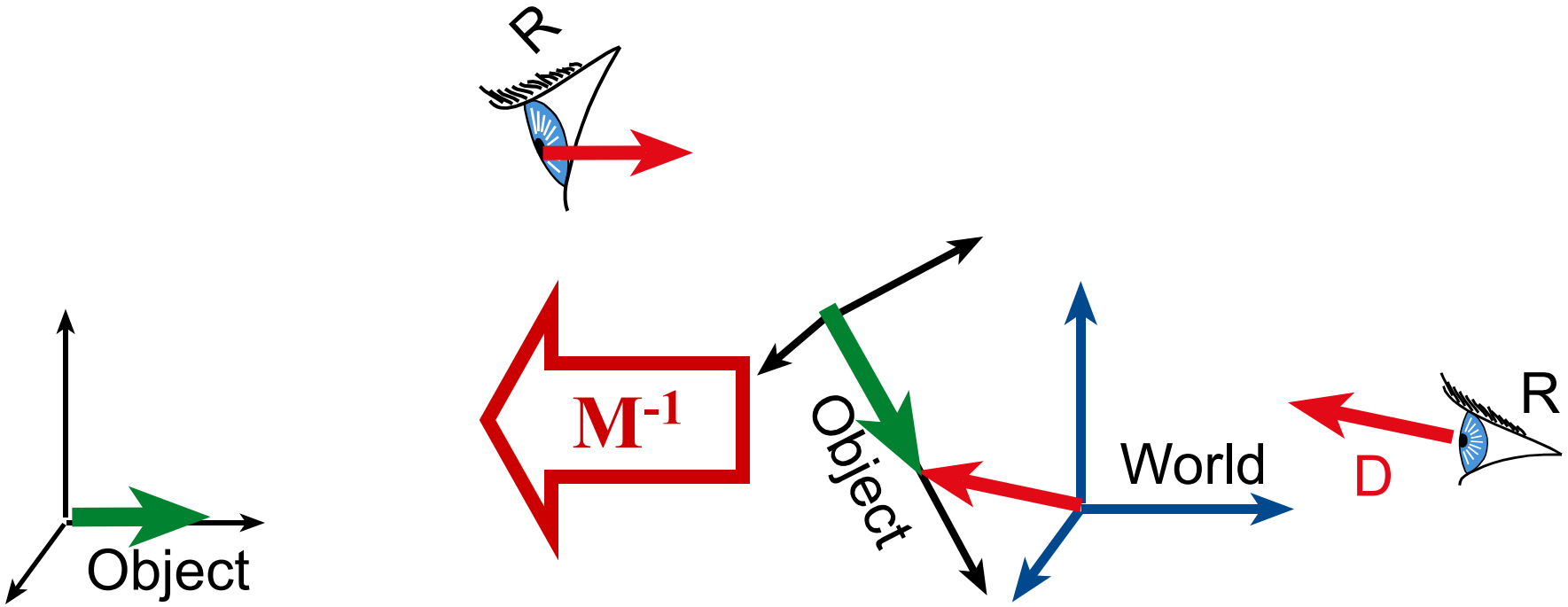
Ray Transformation

- How do we transform the ray's direction?
  - NOT as $D' = M^{-1}D$



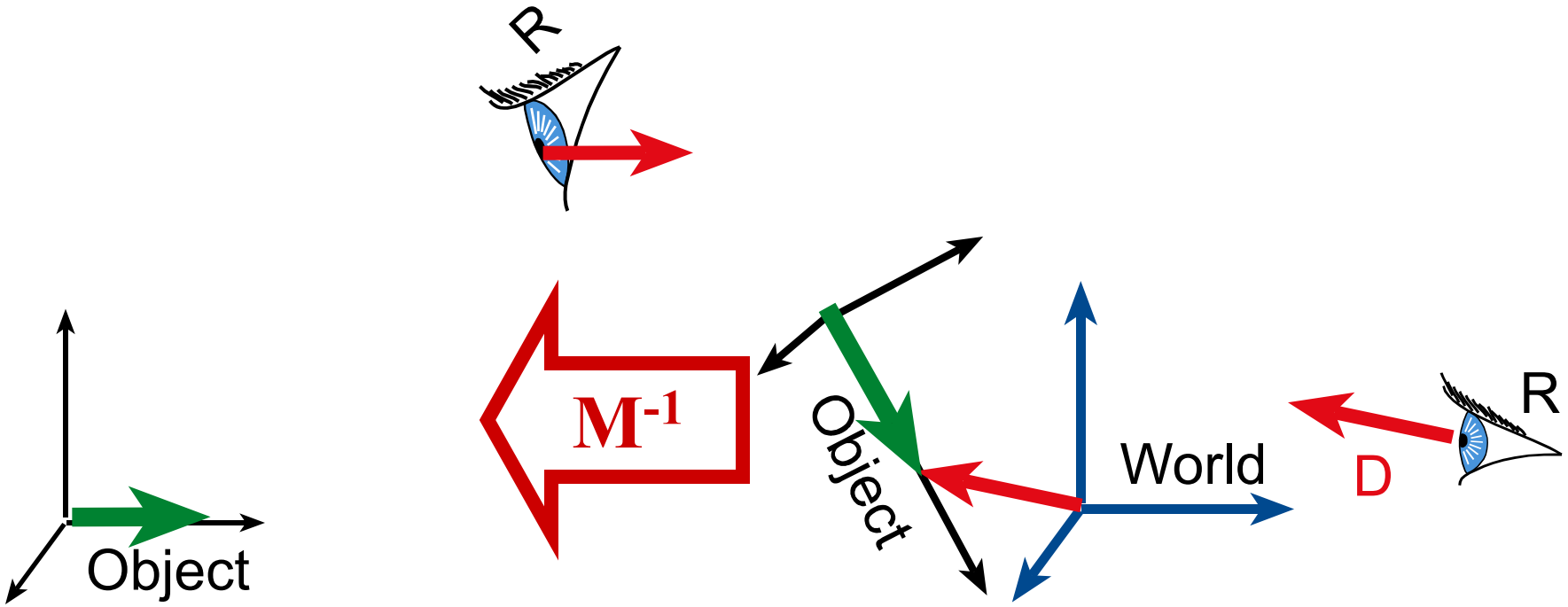
Ray Transformation

- How do we transform the ray's direction?
  - NOT as $D' = M^{-1}D$



Direction vector transformation

- “Endpoints" of ray have equal translations
- Equivalently: ray is a pure direction
- So: transform both ends, subtract results



Direction vector transformation

- Equivalent to ignoring translation part of M

$$\mathbf{M} = \begin{pmatrix} M_{11} & M_{12} & M_{13} & T_x \\ M_{21} & M_{22} & M_{23} & T_y \\ M_{31} & M_{32} & M_{33} & T_z \\ M_{41} & M_{42} & M_{43} & M_4 \end{pmatrix}$$

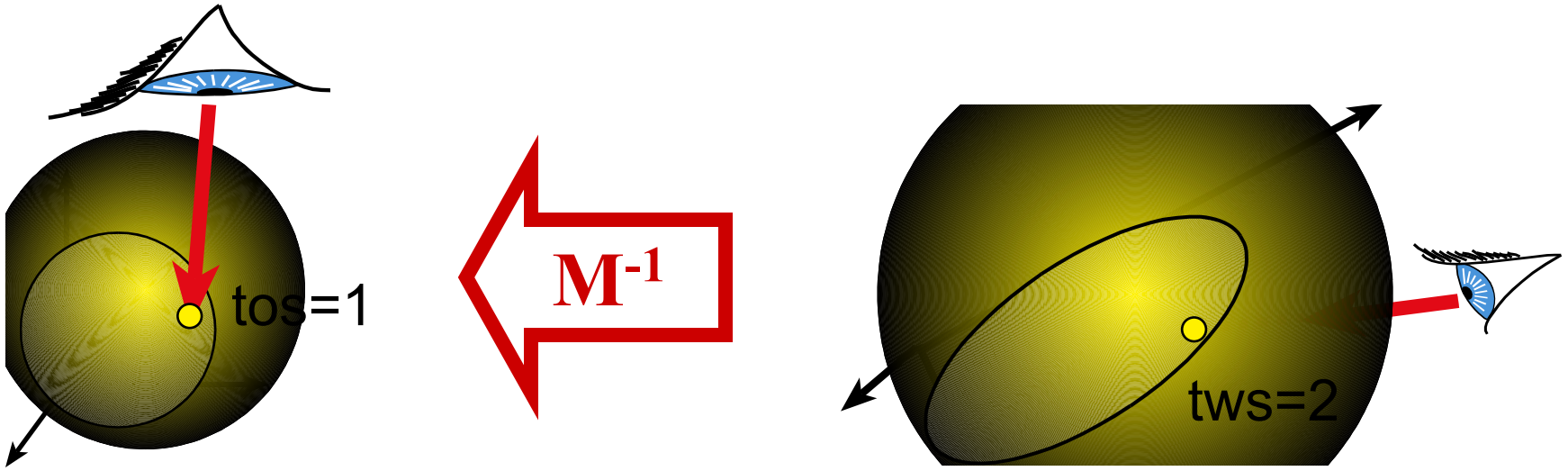
- Instead, use

$$\mathbf{M}^* = \begin{pmatrix} M_{11} & M_{12} & M_{13} & \mathbf{0} \\ M_{21} & M_{22} & M_{23} & \mathbf{0} \\ M_{31} & M_{32} & M_{33} & \mathbf{0} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{pmatrix}$$

- Again: can treat direction (a; b; c) as homogeneous (a; b; c; 0)
- Inventor supplies this as M.multDirMatrix (v; v')

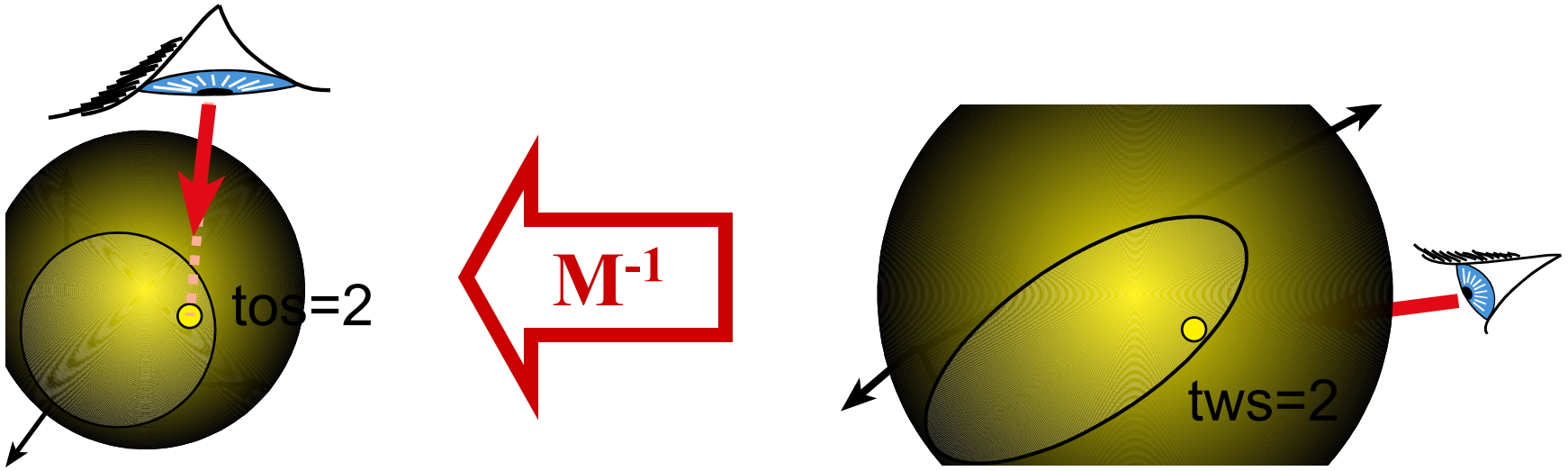
Ray Scaling - Transforming t

- Problem: ray might be scaled anisotropically by M
  - Ok; can still compute intersections in object space
  - How to get t_{WS} from t_{OS}?
- Two options



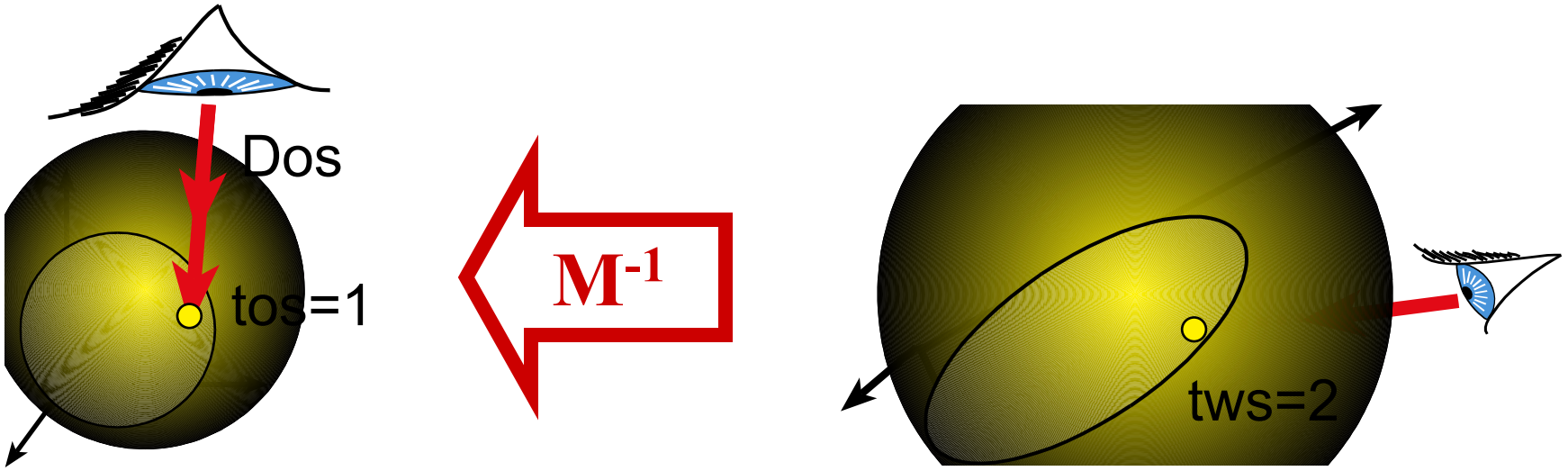
Transforming t - Option 1

- Don't normalize Dos; compute tos
  - Then tws = tos !
  - Must handle unnormalized vectors, but transform easy



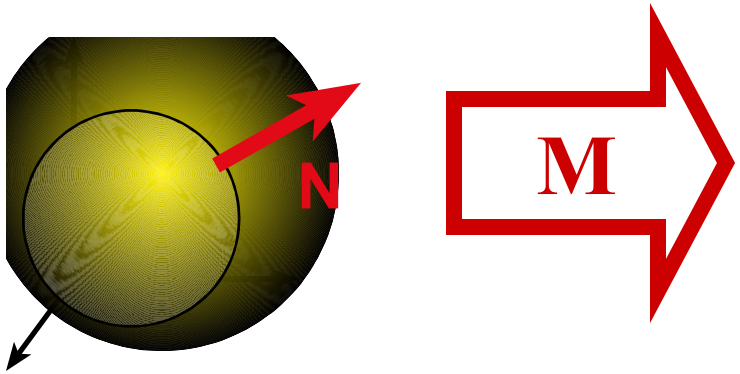
Transforming t - Option 2

- Normalize Dos
- Compute tOS
  - Then tws = tos / ||Dos||
    (||Dos|| before normalization of course)
  - Handle only normalized vectors, but transform harder



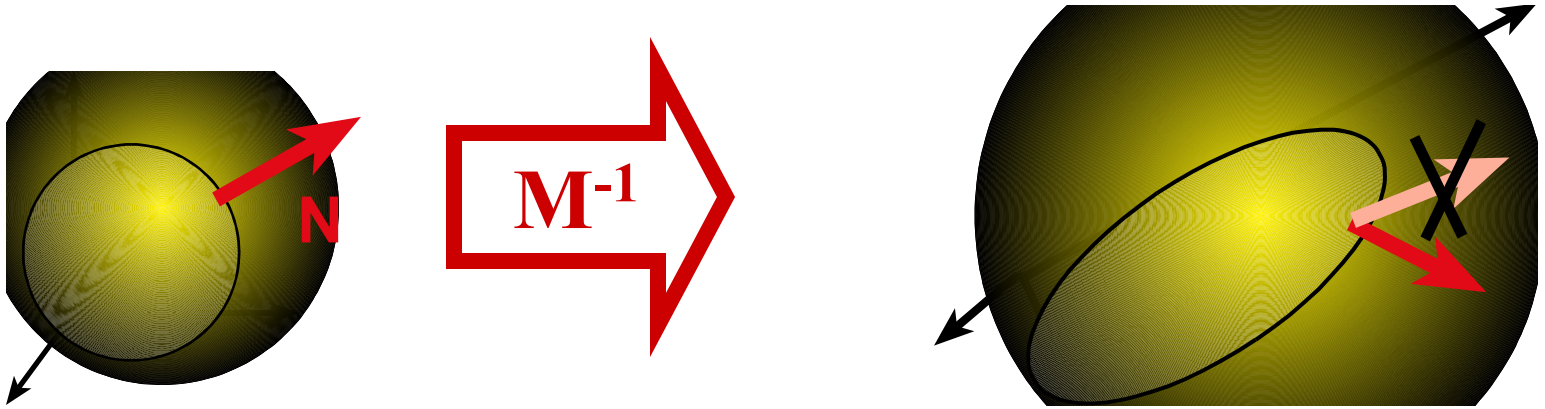
Normal transformation

- How do we transform the normal back to world space?



Normal transformation

- How do we transform the normal back to world space?
- NOT using M
- Using NM^{-1}
  (because if N.V=0, then $(NM^{-1})(MV)=0$)

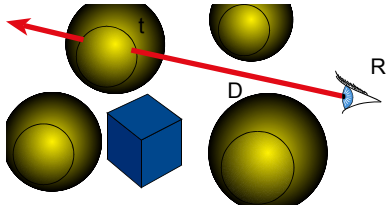


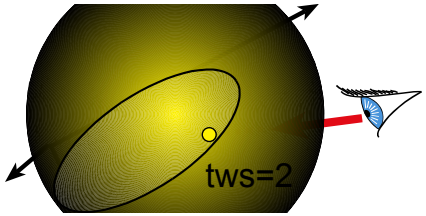
Questions?

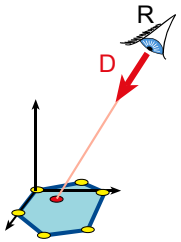
- Image computed using the Dali ray tracer from Henrik Wann jensen
- Model Stephen Duck

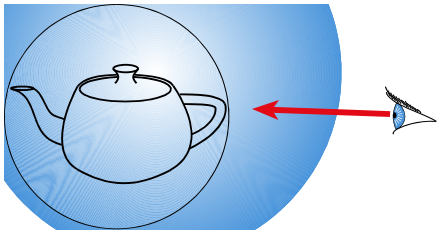
Overview

- Ray casting
- Transformations
- Ray-Primitive intersection
- Advanced ray casting



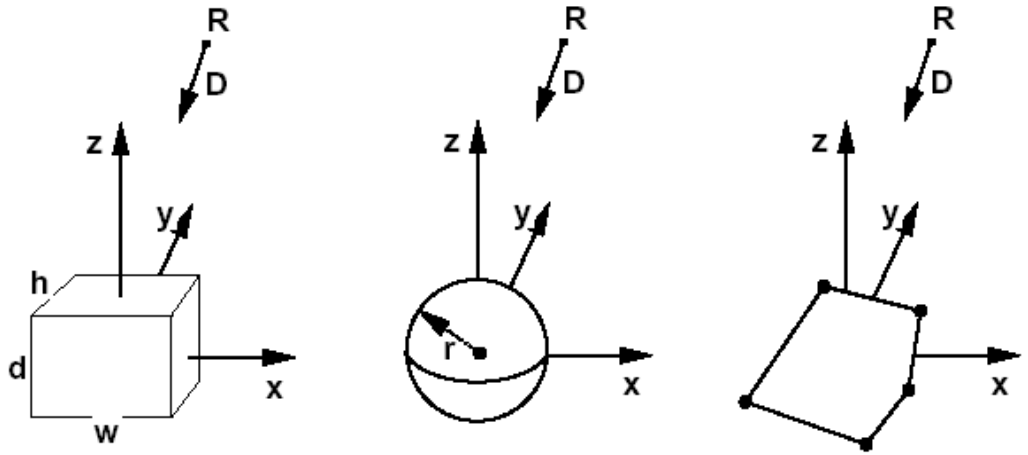






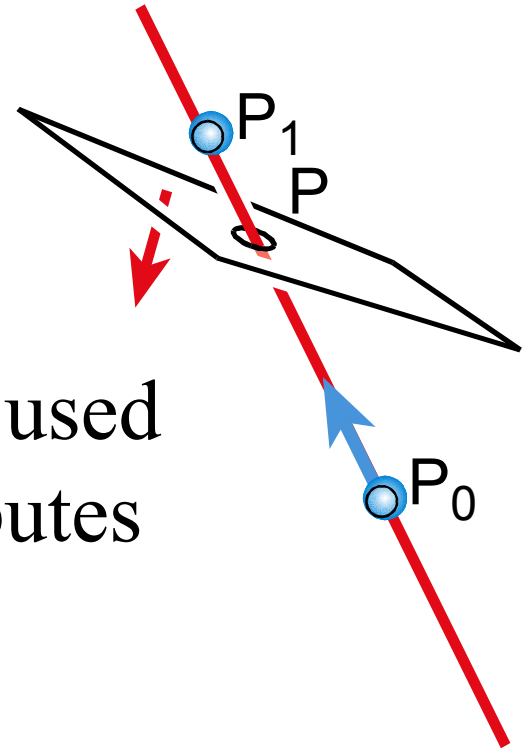
Primitive Intersection

- Now we can work in object space, then revert to world space to return values
- Wish to compute closest intersection point, and normal at that point (if any intersection)
- Consider ray-primitive intersection
- Axial parallelepiped; sphere of radius r; polygon



Line-plane intersection

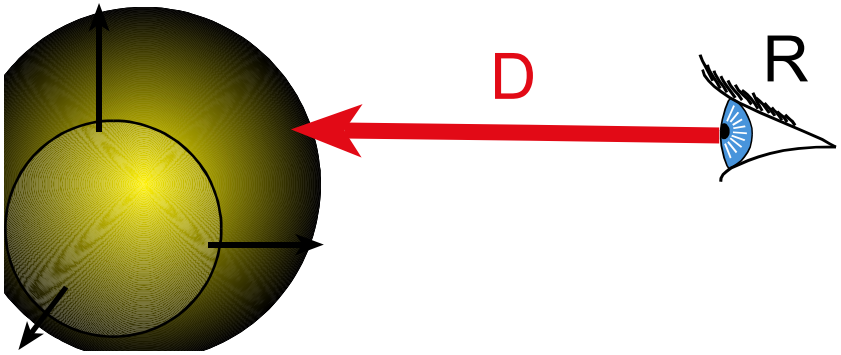
- Insert explicit equation of line into implicit equation of plane



- Parameter *t* can be used to interpolate attributes (color, etc.)

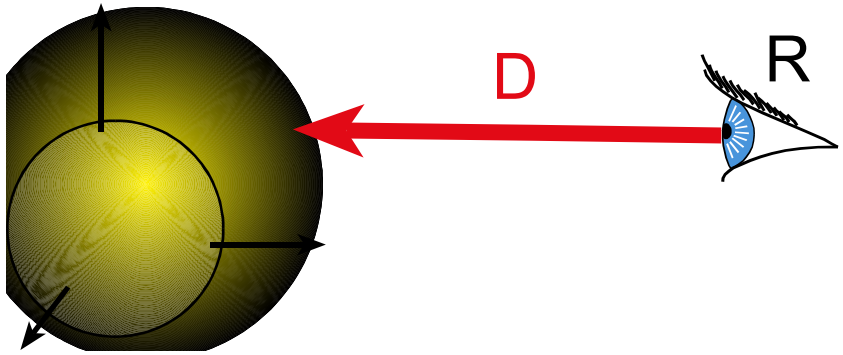
Ray-Sphere Intersection

- Ray equation (explicit): $P(t) = R+tD$ with $||D|| = 1$
- Sphere equation (implicit): $||P||^2 = r^2$
- Intersection means both are satisfied



Ray-Sphere Intersection

$$
\begin{aligned}
0 &= \mathbf{P} \cdot \mathbf{P} - r^2 \\
&= (\mathbf{R} + t\mathbf{D}) \cdot (\mathbf{R} + t\mathbf{D}) - r^2 \\
&= \mathbf{R} \cdot \mathbf{R} + 2t\mathbf{D} \cdot \mathbf{R} + t^2\mathbf{D}^2 - r^2 \\
&= t^2 + 2t\mathbf{D} \cdot \mathbf{R} + \mathbf{R} \cdot \mathbf{R} - r^2
\end{aligned}
$$



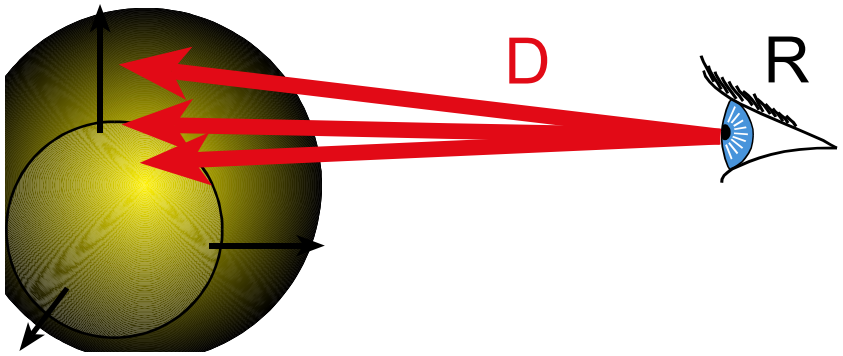
Ray-Sphere Intersection

- This is just a quadratic $at^2 + bt + c = 0$, where
  - a = 1
  - b = 2D.R
  - c = R.R – r^2
- With discriminant $d = \sqrt{b^2 - 4ac}$
- and solutions $t_{\pm} = \frac{-b \pm d}{2a}$

Ray-Sphere Intersection

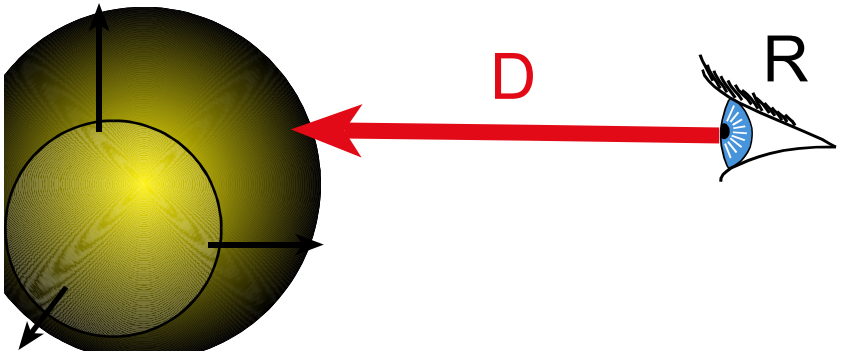
- Discriminant $d = \sqrt{b^2 - 4ac}$
- Solutions $t_{\pm} = \frac{-b \pm d}{2a}$

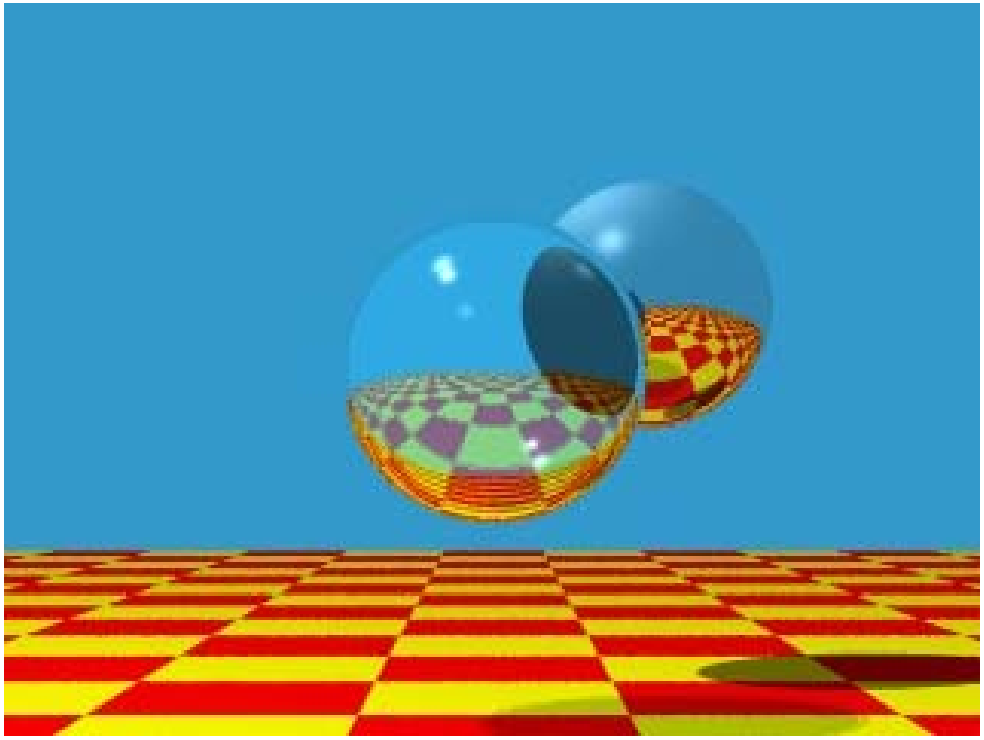
- Three cases, depending on sign of $b^2 - 4ac$
- Which root (t+ or t-) should you choose?



Ray-Sphere Intersection

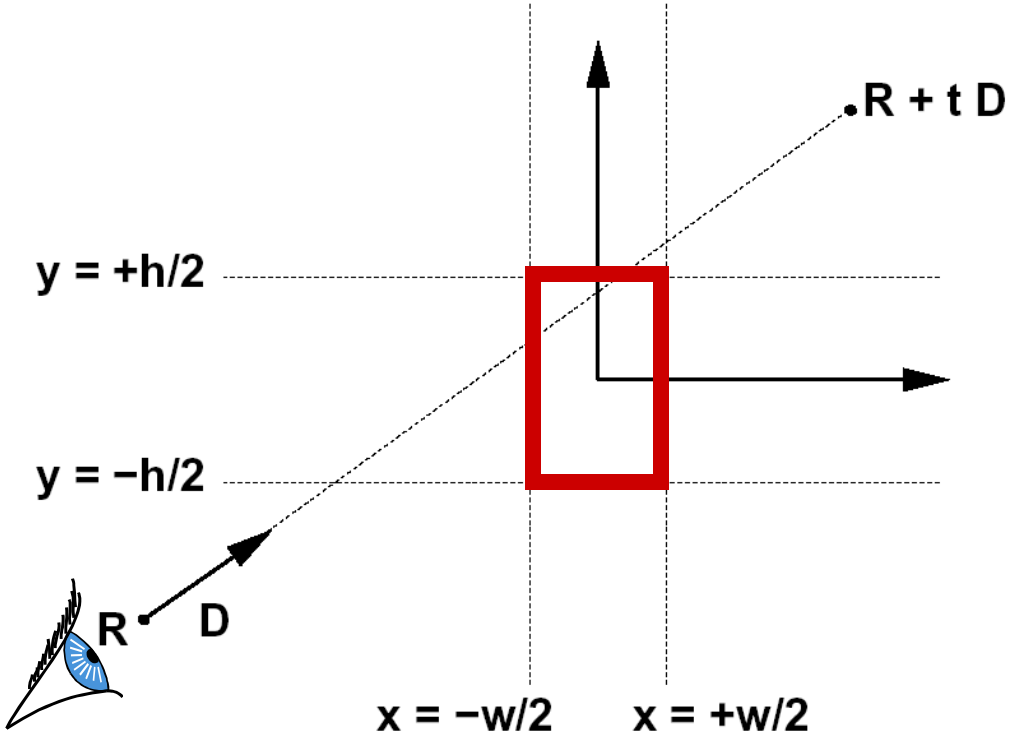
- So easy that all ray-tracing images have spheres!





Ray-Parallelepiped Intersection

- Intersect with all 6 planes
- Clip intersections
- Keep tmin

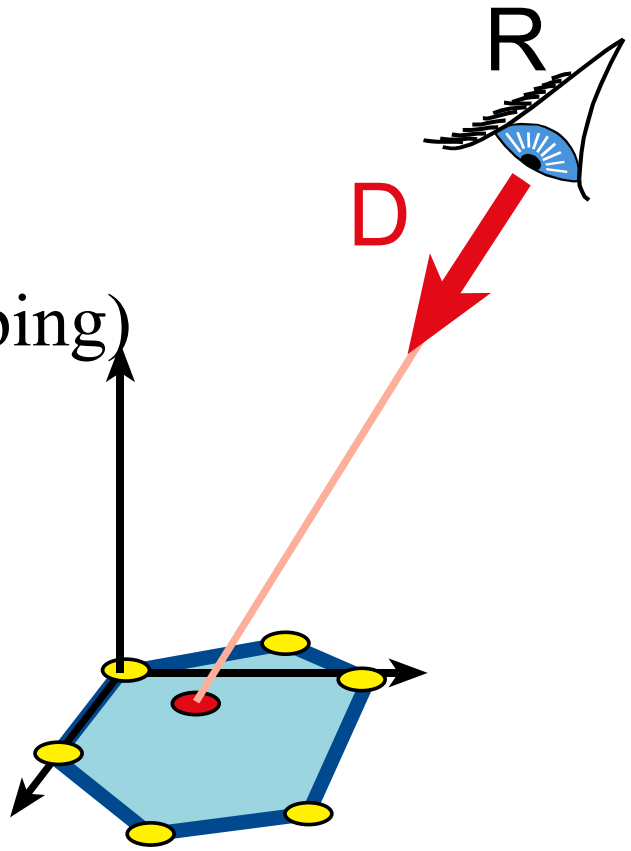


Ray-Parallelepiped Intersection

```
static Vector hats[3] = { xhat, yhat, zhat };
// ray is of form R + t D; assign min t as thit; normal N
float t1, t2, tmin = 0, tmax = HUGE; // from <math.h>
Vector extent = Vector ( width / 2, height / 2, depth / 2 );
// intersect ray with x, y, z ``slabs'' (k = 0, 1, 2)
for ( int k = 0; k < 3; k++ ) {
   if ( D[k] != 0. ) {
        t1 = (-extent[k] - R[k]) / D[k]; // plane x_k = -dx_k
        t2 = ( extent[k] - R[k]) / D[k]; // plane x_k = +dx_k
        tmin = fmax ( tmin, fmin (t1, t2)); // intersect [tmin..
        tmax = fmin ( tmax, fmax (t1, t2)); // tmax], [t1..t2]
        if ( tmax <= tmin ) return FALSE; // no intersection
        else    if ( tmin == t1 ) N = -hats[k]; // hit x_k = -dx_k
                else if ( tmin == t2 ) N = hats[k];// hit x_k = +dx_k
        } // if
} // k
thit = tmin; // return parameter of closest intersection
return TRUE;
```

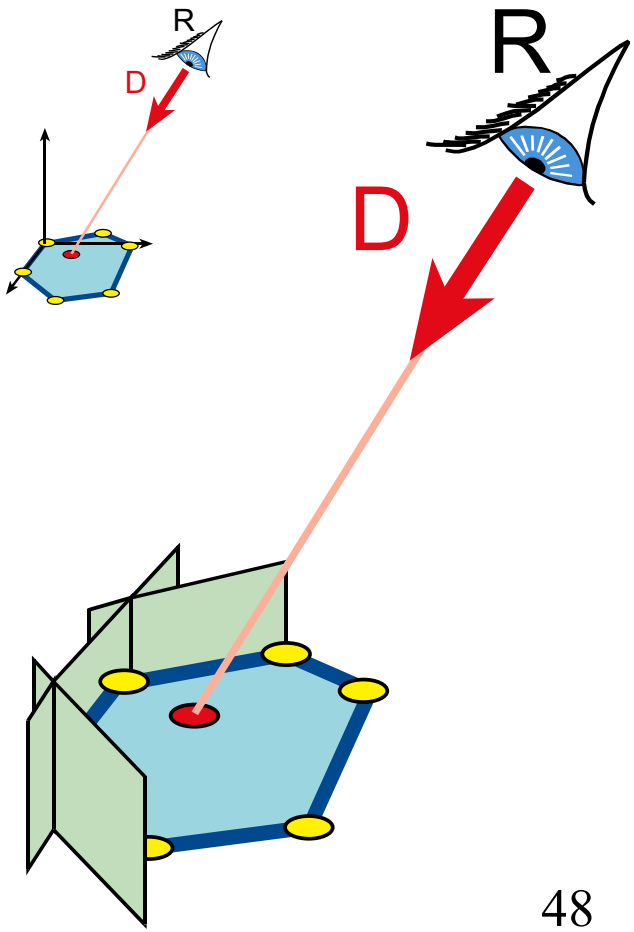
Ray-Polygon Intersection

- Options:
- Transform to polygon space (polygon plane become xy)
  - Transform ray
  - Intersect ray with (xy)
  - Solve 2D inclusion problem (clipping)



Ray-Polygon Intersection

- Options:
- Transform to polygon space (polygon plane become xy)
  - Transform ray
  - Intersect ray with (xy)
  - Solve 2D inclusion problem (clipping)
- Intersect ray with support plane
  - Compute intersection point
  - Express 3D edges as plane constraints
  - Solve 3D inclusion problem
- Tradeoffs?

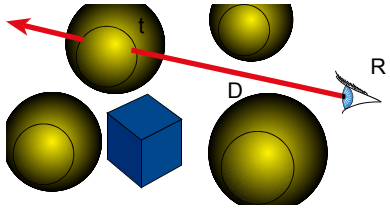


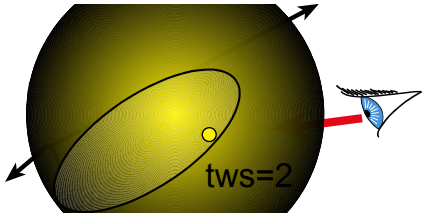
Questions?

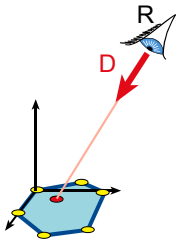
- Image computed using the Dali ray tracer from Henrik Wann jensen
- Environment map by Paul Debevec

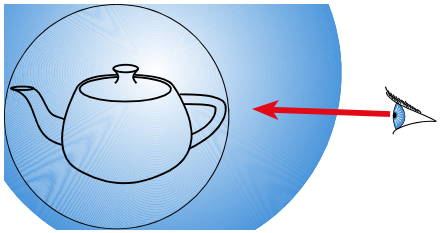
Overview

- Ray casting
- Transformations
- Ray-Primitive intersection
- Advanced ray casting



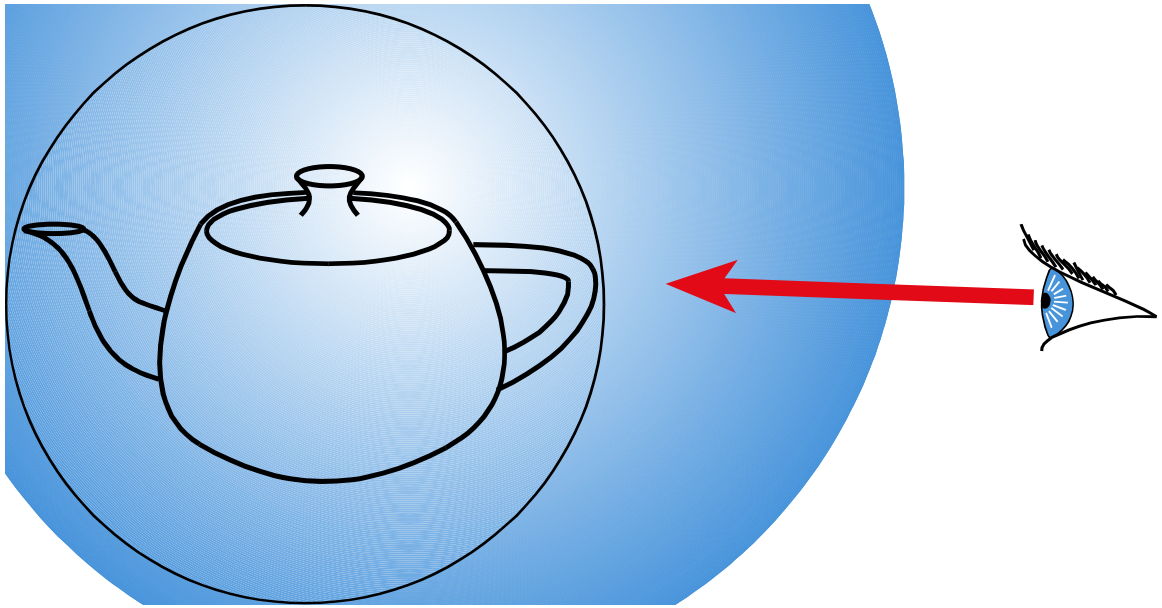




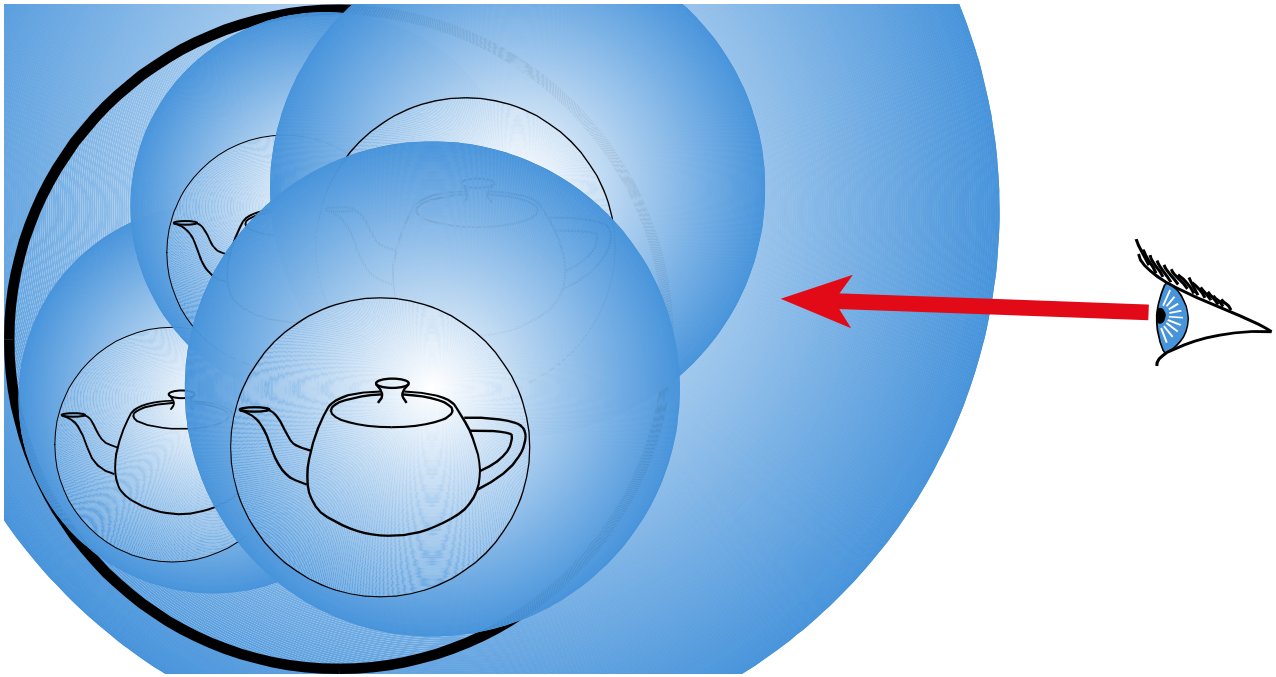


Acceleration: Bounding volumes

- Simpler volume that encloses the object
- If bounding volume not intersected, early rejection
- Tradeoff simplicity of intersection/tightness

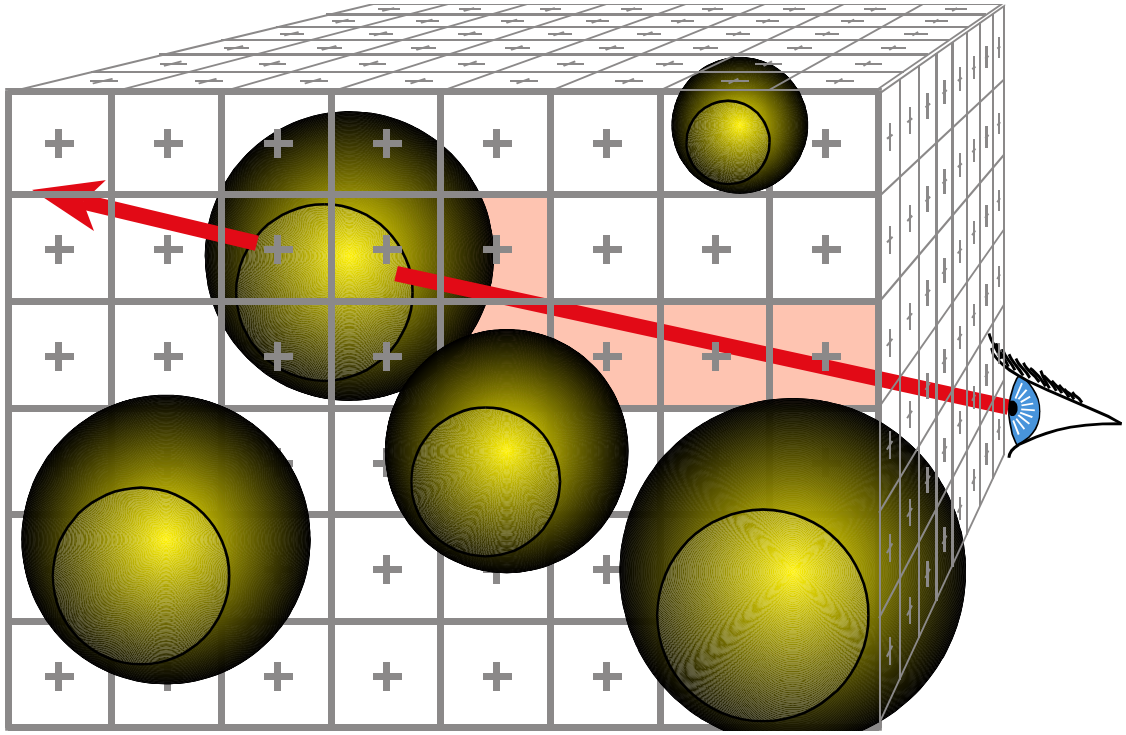


Hierarchical Bounding volumes



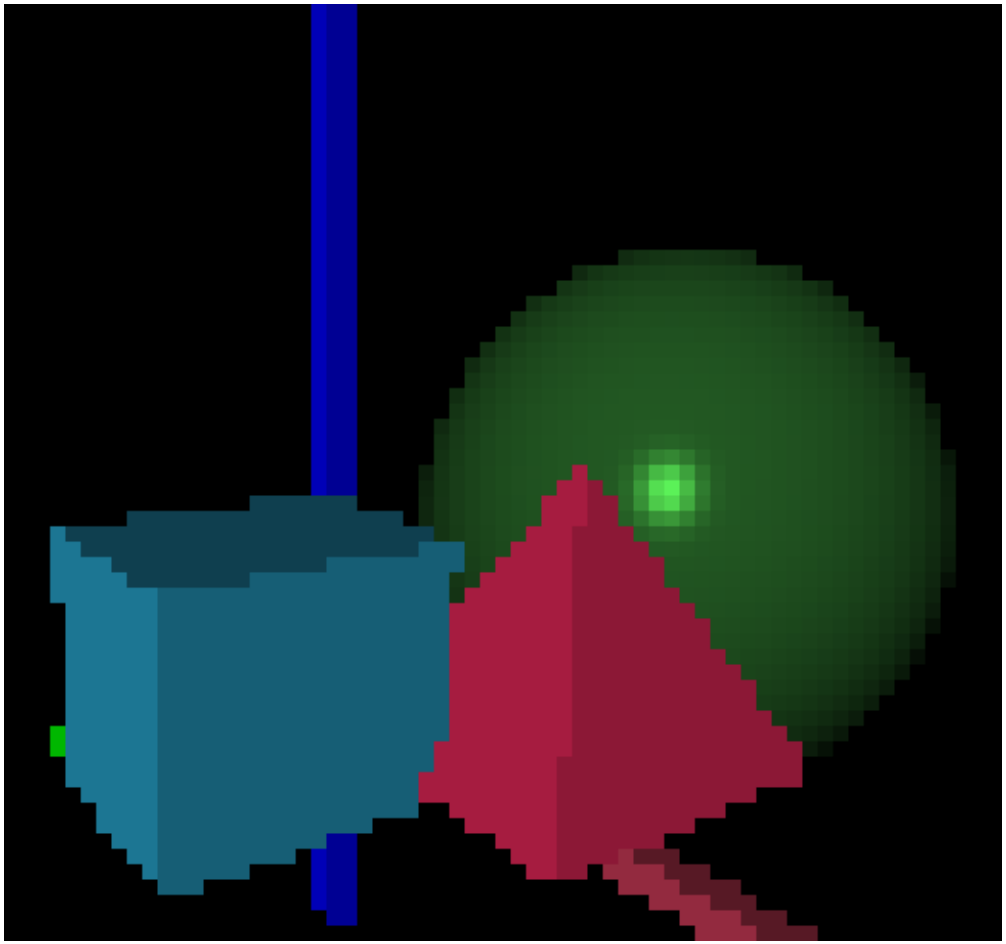
Acceleration: Grid

- Extension of Bresenham
- We must visit all “voxels” intersected by Ray
- Subtleties for objects that span multiple voxels



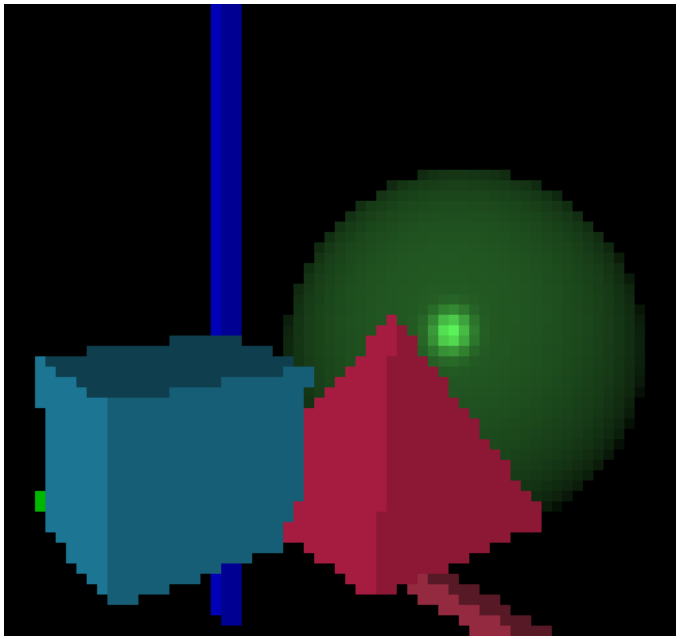
Aliasing

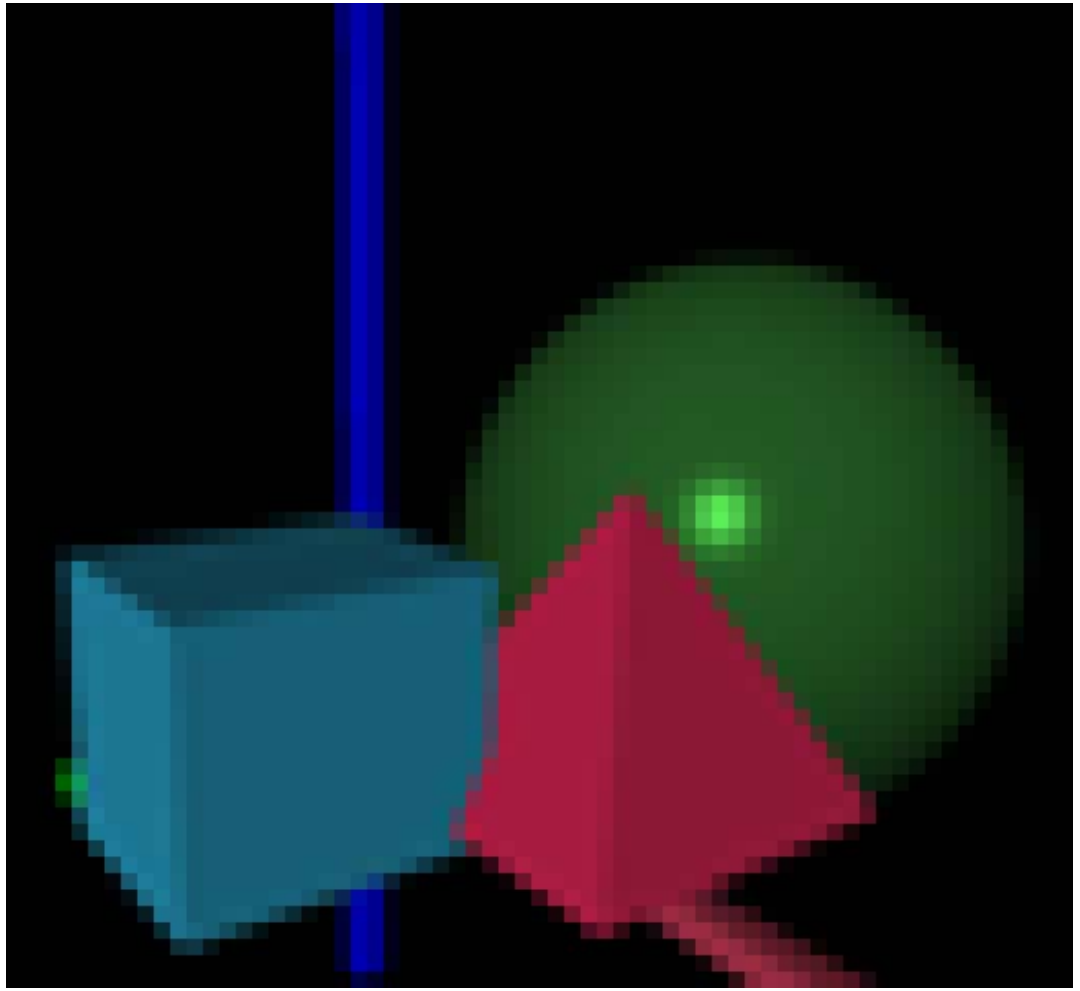
- So far, both rasterization and raycasting compute one color per pixel:
- Look closely at:
- Notice “jaggies” at object boundaries
  - Why do they happen ?
- Can we do better? How?



Super Sampling

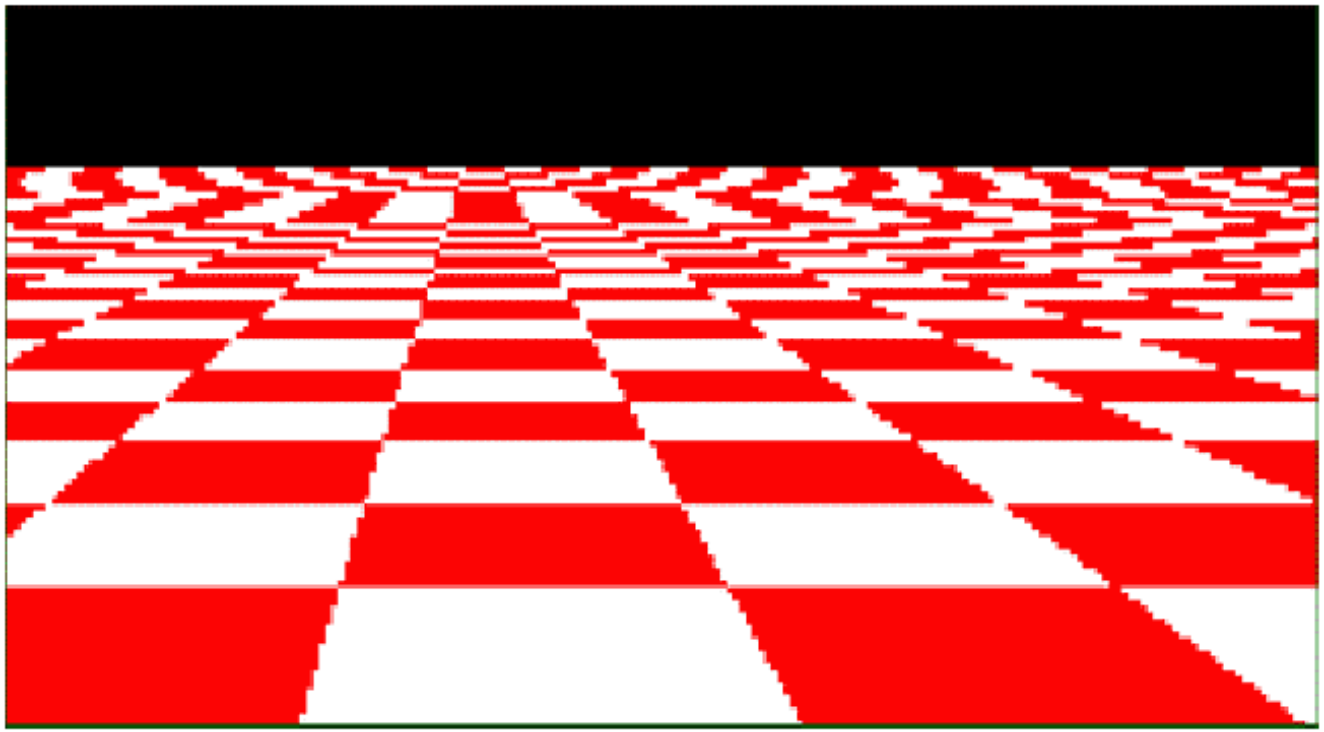
- We can average multiple samples per pixel:





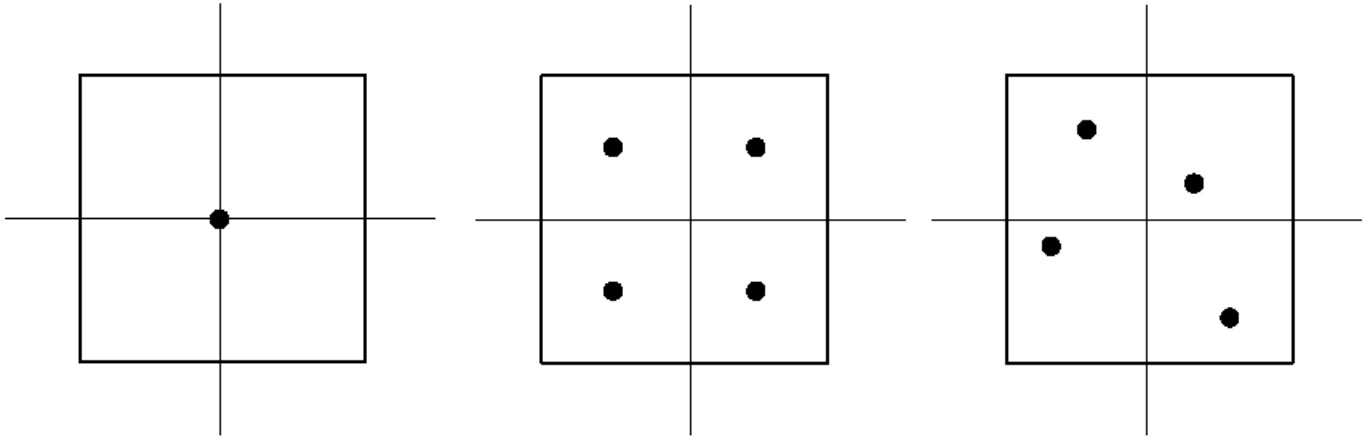
More Antialiasing

- Important when rendering high-frequency textures
- More about this in November



Placing Samples

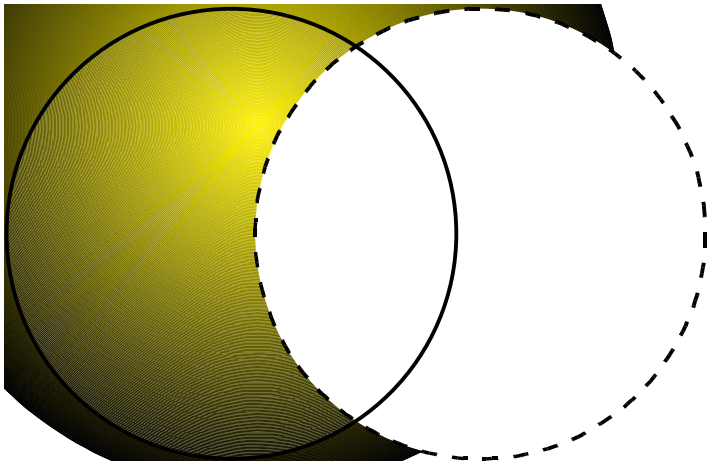
- Regular, Jittered Supersampling

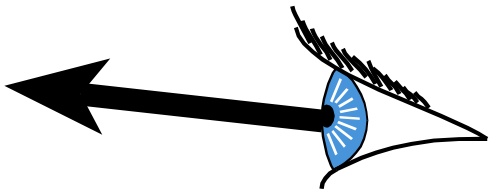


- Noise is better than aliasing

Constructive Solid Geometry (CSG)

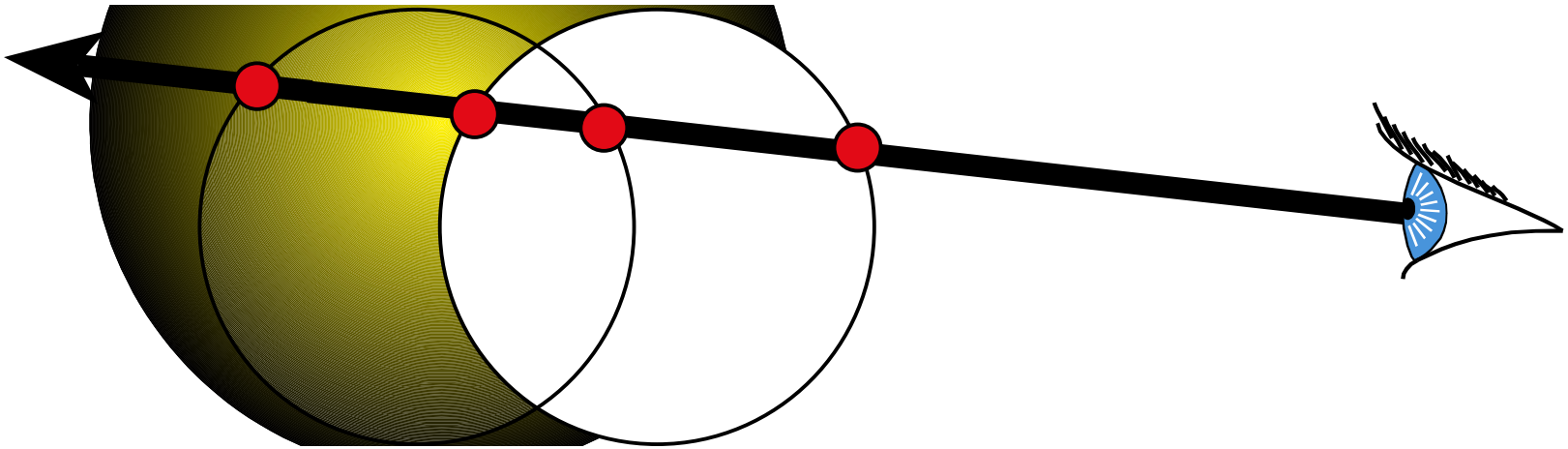
- Simple with some rendering algorithms (Ray Tracing)



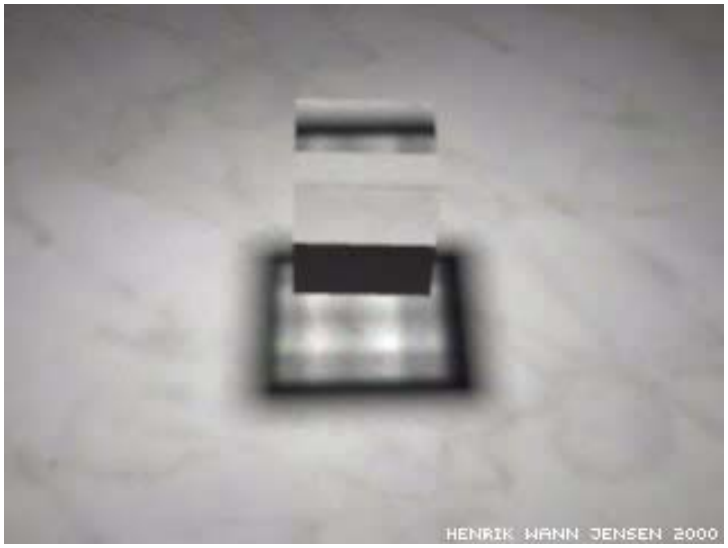


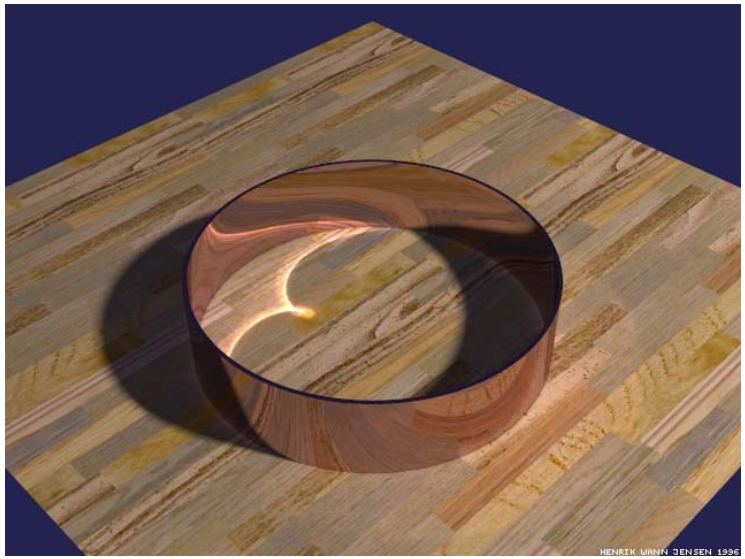
Constructive Solid Geometry (CSG)

- Simple with some rendering algorithms (Ray Tracing)



Thursday: Ray Tracing





Images and animations by Henrik Wann Jensen