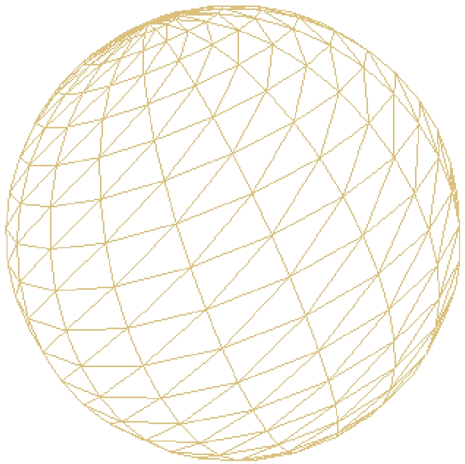
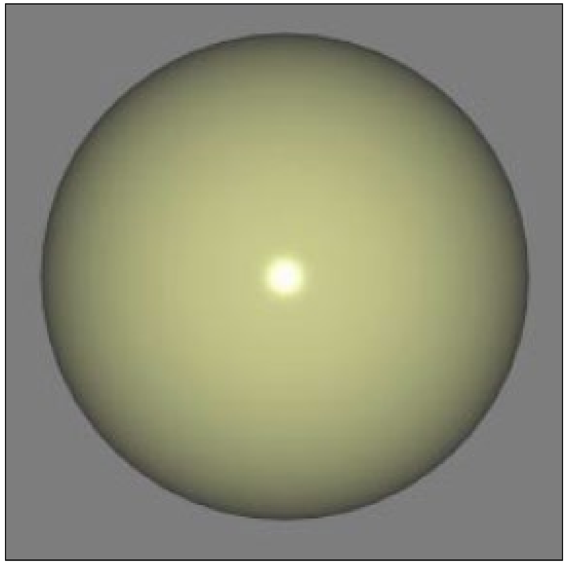
Rendering



Model / scene comprised of geometric primitives in 3D coordinate space

Rendering

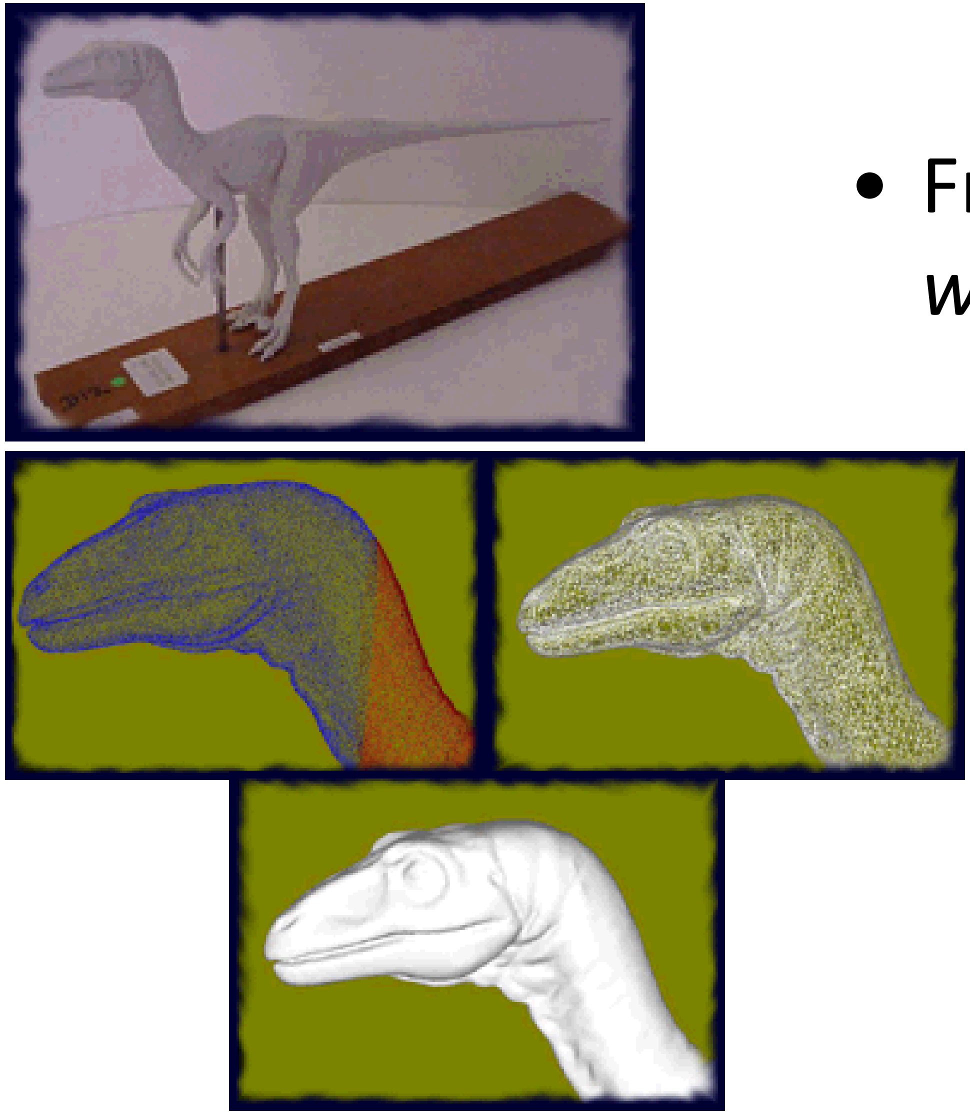
Transformation of 3D space

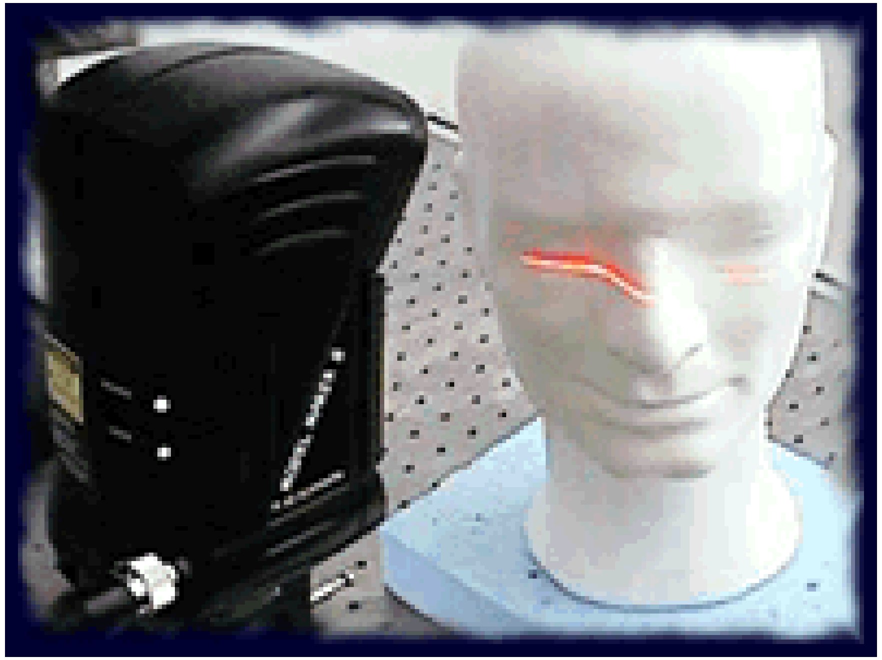


Raster image

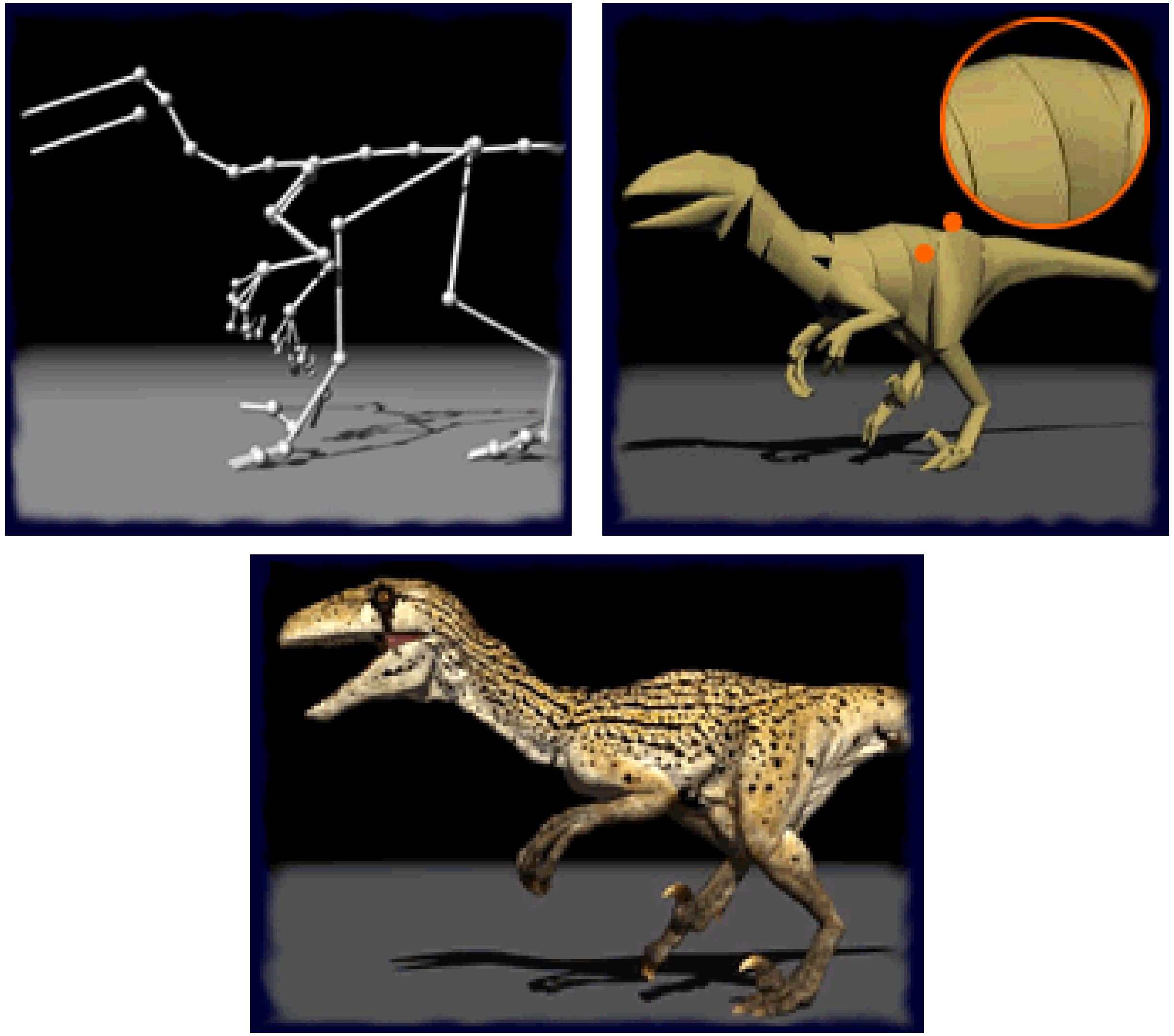
Modelling via capture

- Framestore's *Walking with Dinosaurs*

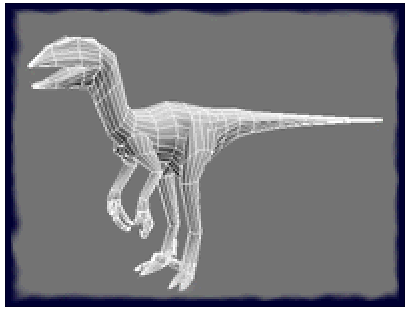


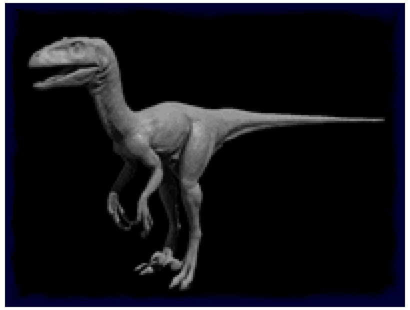


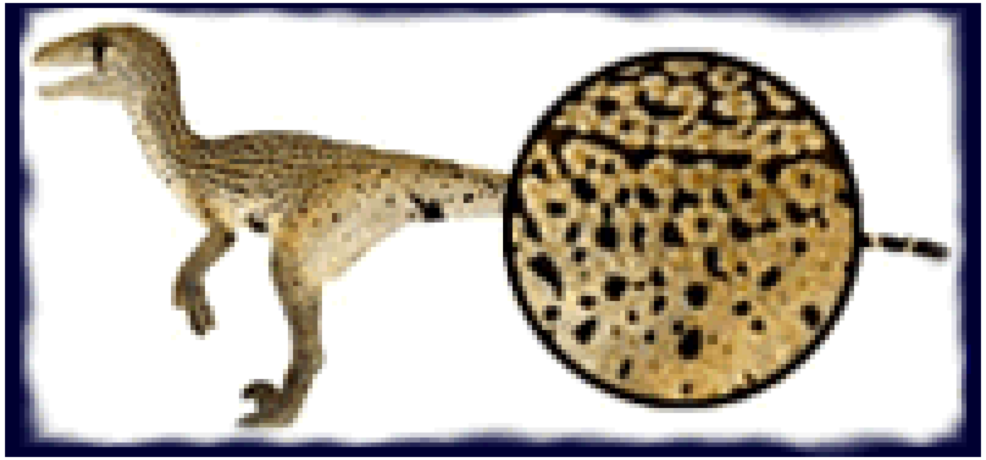
Animation



Textures







Lighting

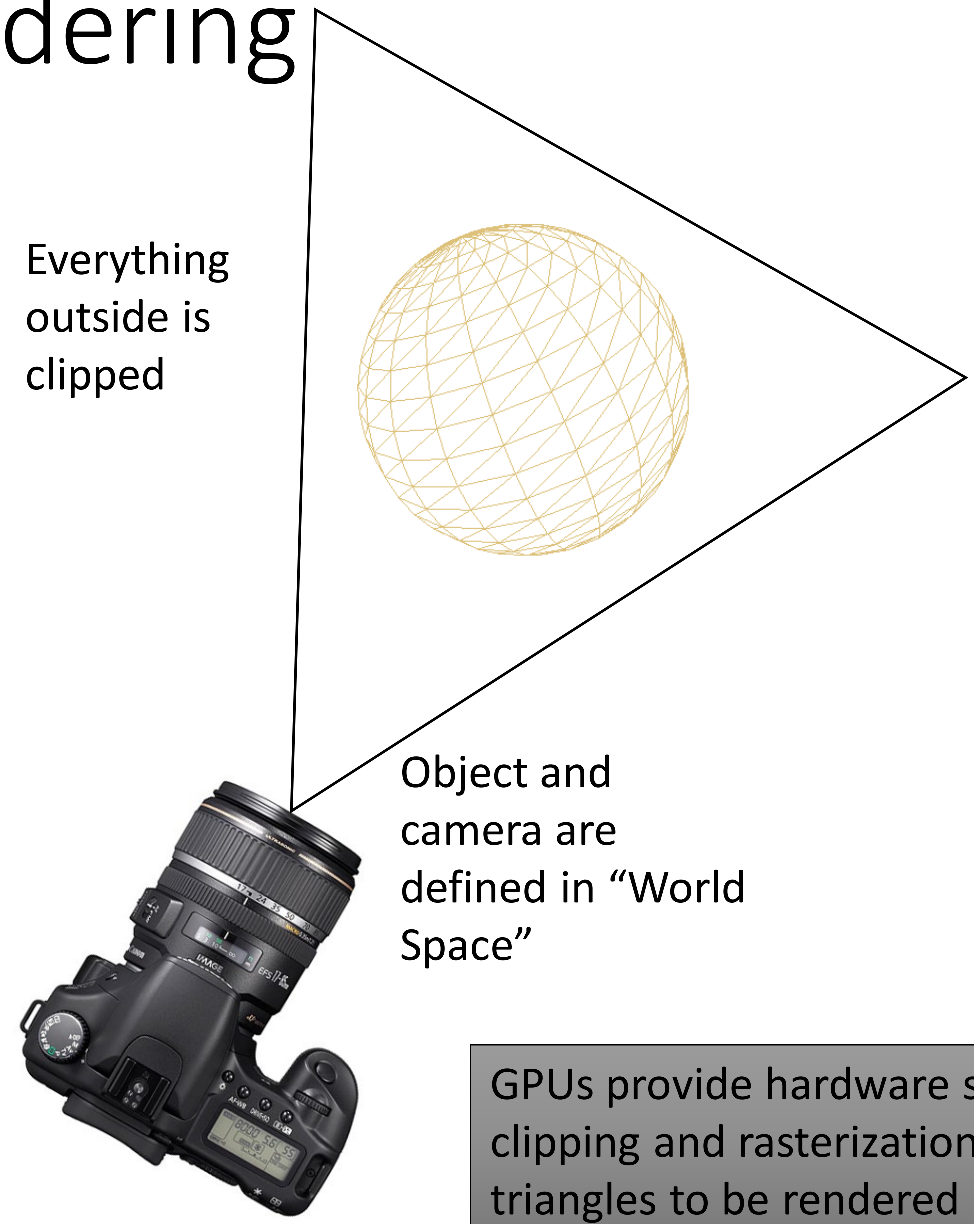
- Range
- Light source and intensity

Rendering

- We will study two forms of rendering – rasterization (briefly) and ray tracing (in detail)
- **Rasterization** consists of several steps:
  - **Transformation**, **clipping** and **scan conversion**
- **Matrices** for scaling, translation and rotation are applied to each vertex within the object during transformation
- After transformation each vertex (x,y) gives the screen coordinate, and z gives the depth
- **Clipping** removes any part of the scene not visible within the image
- Scan conversion colours pixels according to the object's colour and the **lighting model**

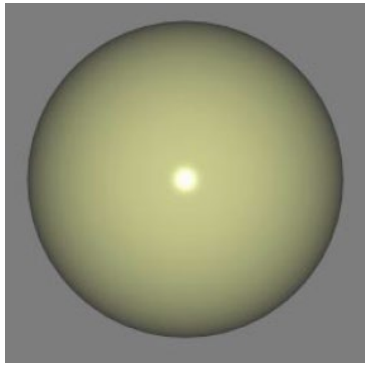
Rendering

Everything outside is clipped



Object and camera are defined in “World Space”

World coordinates are transformed into view coordinates (x,y,z)

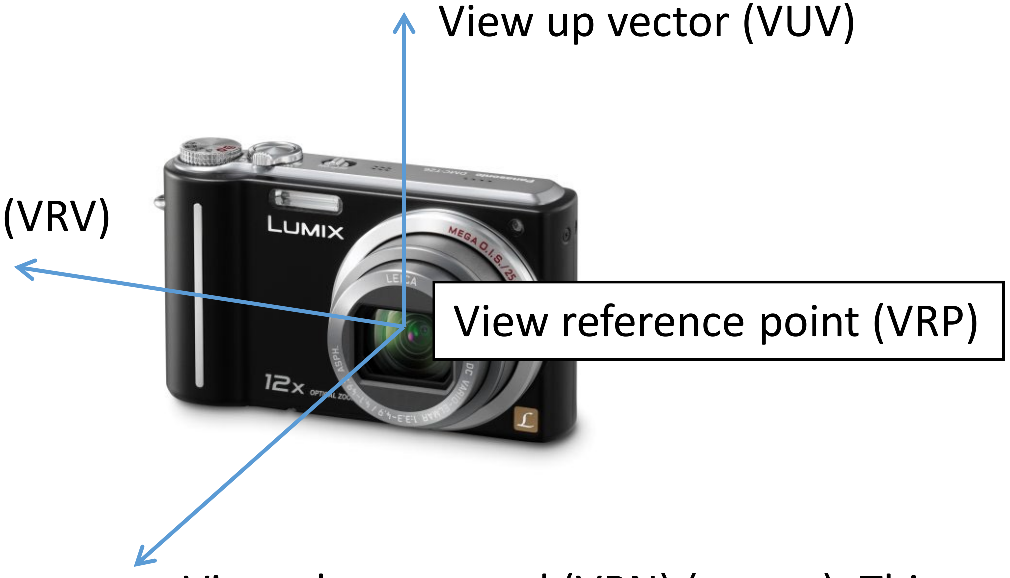


Such that (x,y) give the position within the view (image) and z gives the depth to that position. The depth can be used to make sure that occluded surfaces are hidden by closer surfaces

GPUs provide hardware support for transformation, clipping and rasterization allowing scenes of millions of triangles to be rendered in real-time

Camera Model

For both Rasterization and Ray Tracing, we need to define a camera model. We explore which parameters are needed

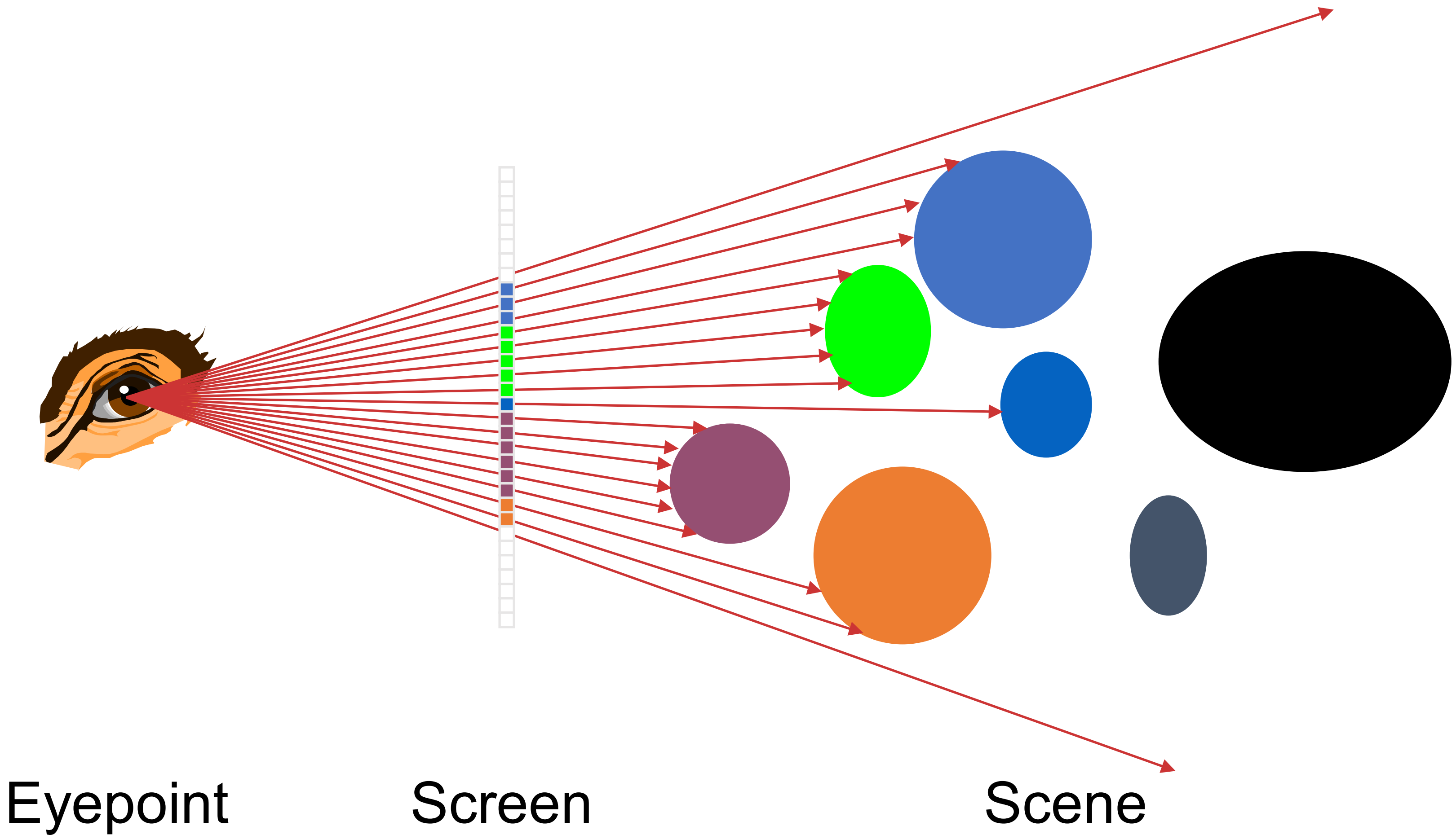


View plane normal (VPN) (vector). This can be calculated from the position (point) the camera looks at minus the view reference point of the camera

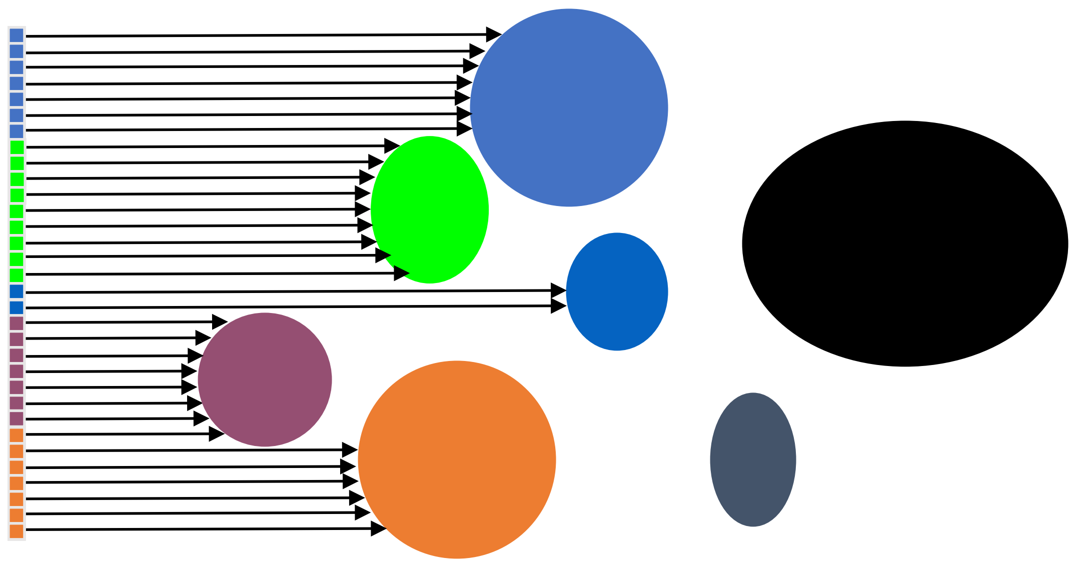
Transformation

- The transformation matrix can be calculated from the view plane normal (=look at – vrp), the view up vector and the view right vector
- For rasterization, each vertex is multiplied by the matrix (in GPU hardware)
- The resulting (x,y,z) points can be clipped and scan converted
- In ray tracing, rays are sent out from the view plane, into the scene to detect which objects are hit
- **Ray tracing** is studied in more detail in the next lectures

Ray Tracing: Perspective Projection



Orthographic Projection

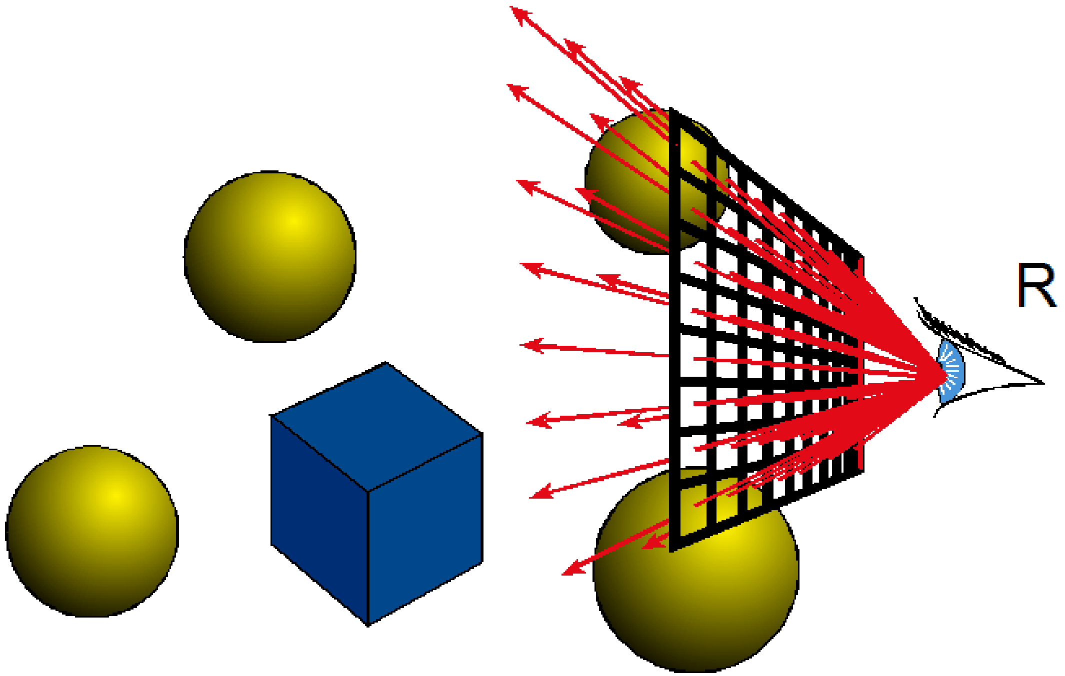


Screen

Scene

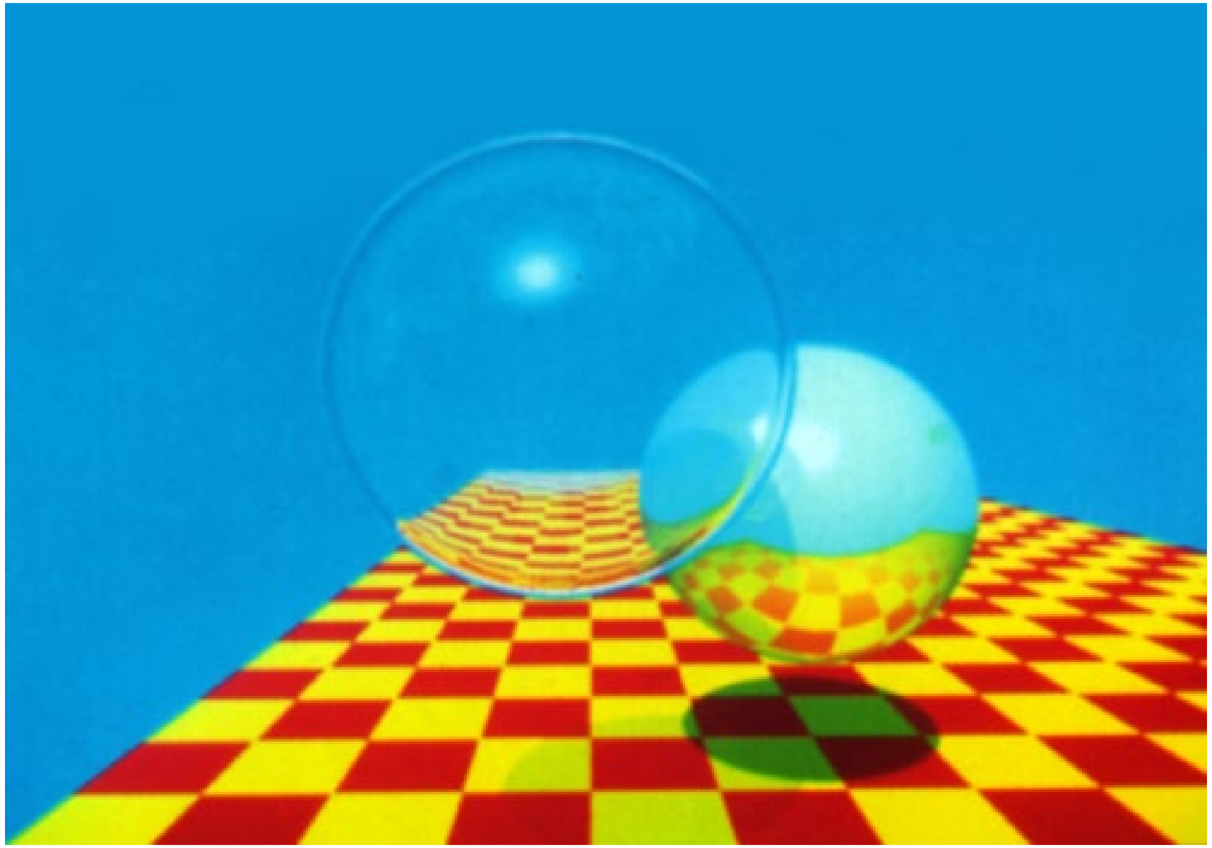
Ray Tracing

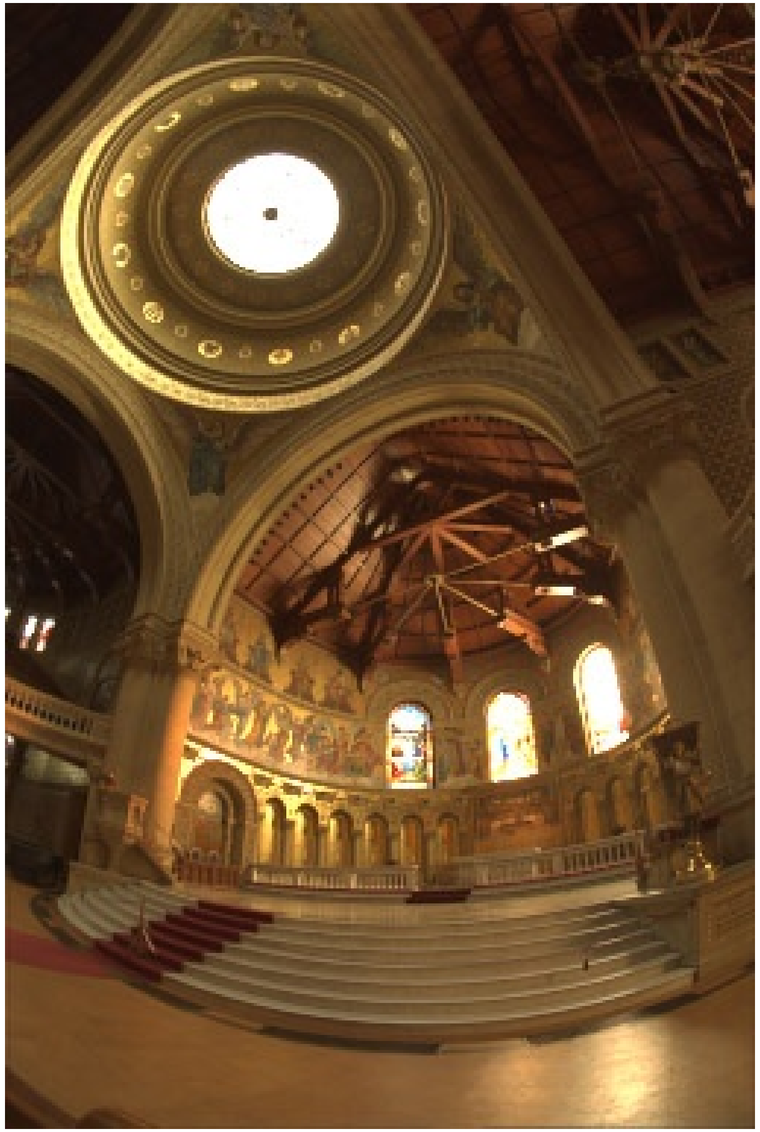
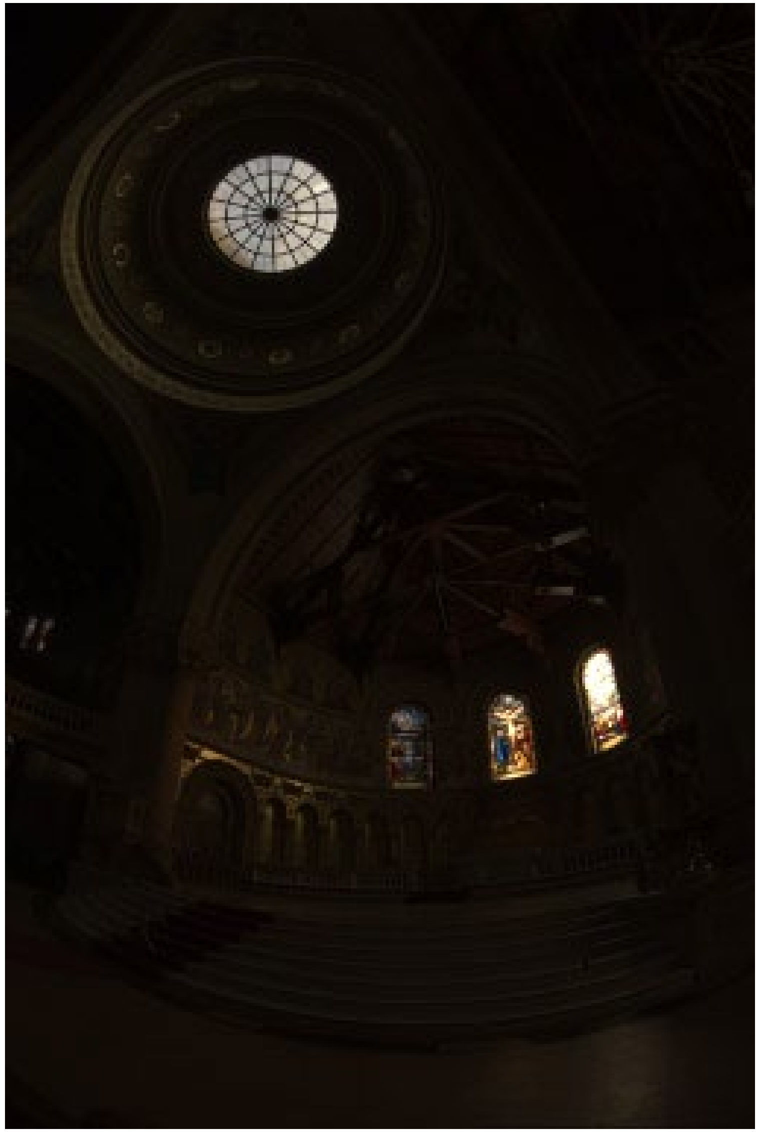
- Similar diagram (but in 3D)



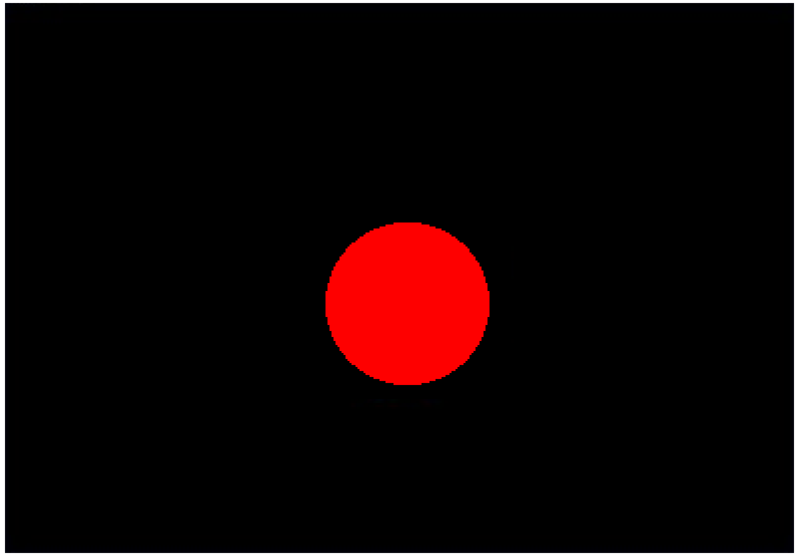
Ray Tracing - 1979

- Shadows, refraction, reflection and texture mapping



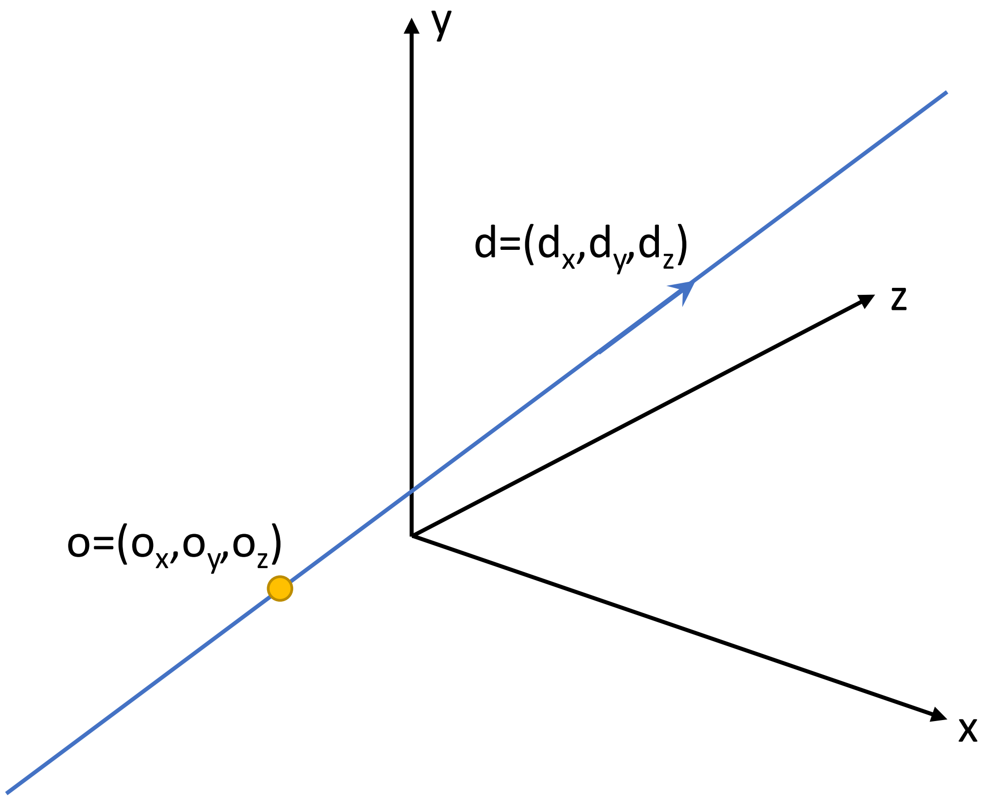







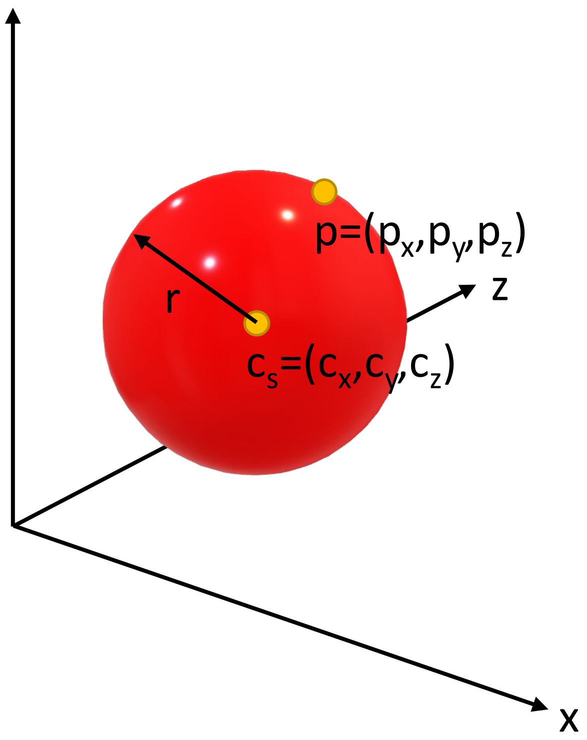
- What about a 3rd year project in ray tracing?
- The first image most people see!
- Intersection between
  3D line $p = o + dt$ and
  sphere $(p - c_s)^2 = r^2$
- o, origin of ray (3D coordinate of pixel)
- d, direction of ray (VPN in orthographic)
- c_s, centre of sphere
- r, radius of sphere
- p, 3D points (on line or sphere)
- t, solution we seek – where on the line it intersects the sphere

Ray Sphere Intersection



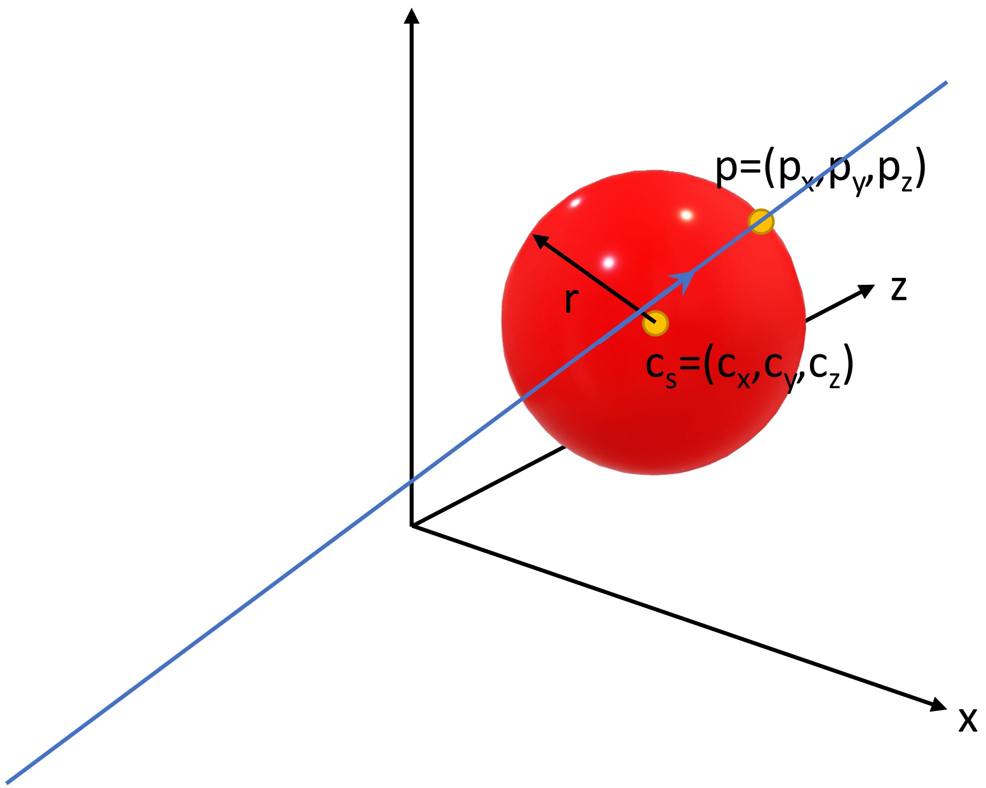
A 3D line is defined
p=o+dt
where
o is the origin of the line (3D point),
d is the direction (3D vector) of the line
t is a scalar where $-\infty \leq t \leq \infty$

Ray Sphere Intersection



A 3D sphere is defined
$(p-c_s)^2=r^2$
where
c_s is the origin of the sphere (3D point),
p is any point on the sphere surface (3D point),
r is a scalar, radius of the sphere

Ray Sphere Intersection



Use the two equations to determine the points of intersection between the ray and sphere

We want to find a point on the line that is also a point on the surface of the sphere

The points on the line are p in $p=o+dt$

The points on the sphere are p in $(p-c_s)^2=r^2$

Ray Sphere Intersection

Line: $p=o+dt$

Sphere: $(p-c_s)^2=r^2$

Substitute $p=o+dt$ from line equation into the sphere equation:

$(p-c_s)^2=r^2$

$\downarrow$

$(o+dt-c_s)^2=r^2$

o is the origin of the line (a 3D point)

c_s is the centre of the sphere (a 3D point)

Set $v=o-c_s$

v is now a vector from the centre of the sphere to the origin of the line

$(o+dt-c_s)^2=r^2$

becomes

$(v+dt)^2=r^2$

Ray Sphere Intersection

$(v+dt)^2=r^2$
Expands to
$v^2+2vdt+d^2t^2=r^2$ (search algebraic manipulation if you are stuck here)
Rearrange to give
$d^2t^2+2vdt+v^2-r^2=0$
This is an equation of the form
$at^2+bt+c=0$ (search quadratic formula if you are stuck here)
where $a=d^2$, $b=2vd$, and $c=v^2-r^2$
Remember d is a 3D vector (direction of line)
Therefore $a=d^2$ is calculated as a=d dot product d, which is
$a = d_x * d_x + d_y * d_y + d_z * d_z$ (so a is a scalar) (search dot product if you are stuck here)

Ray Sphere Intersection

$a=d^2$, $b=2vd$, and $c=v^2-r^2$

Similarly

$b=2*(v_x * d_x + v_y * d_y + v_z * d_z)$

and

$c=(v_x * v_x + v_y * v_y + v_z * v_z)-r^2$

Ray Sphere Intersection

For quadratic formula of the form $at^2+bt+c=0$

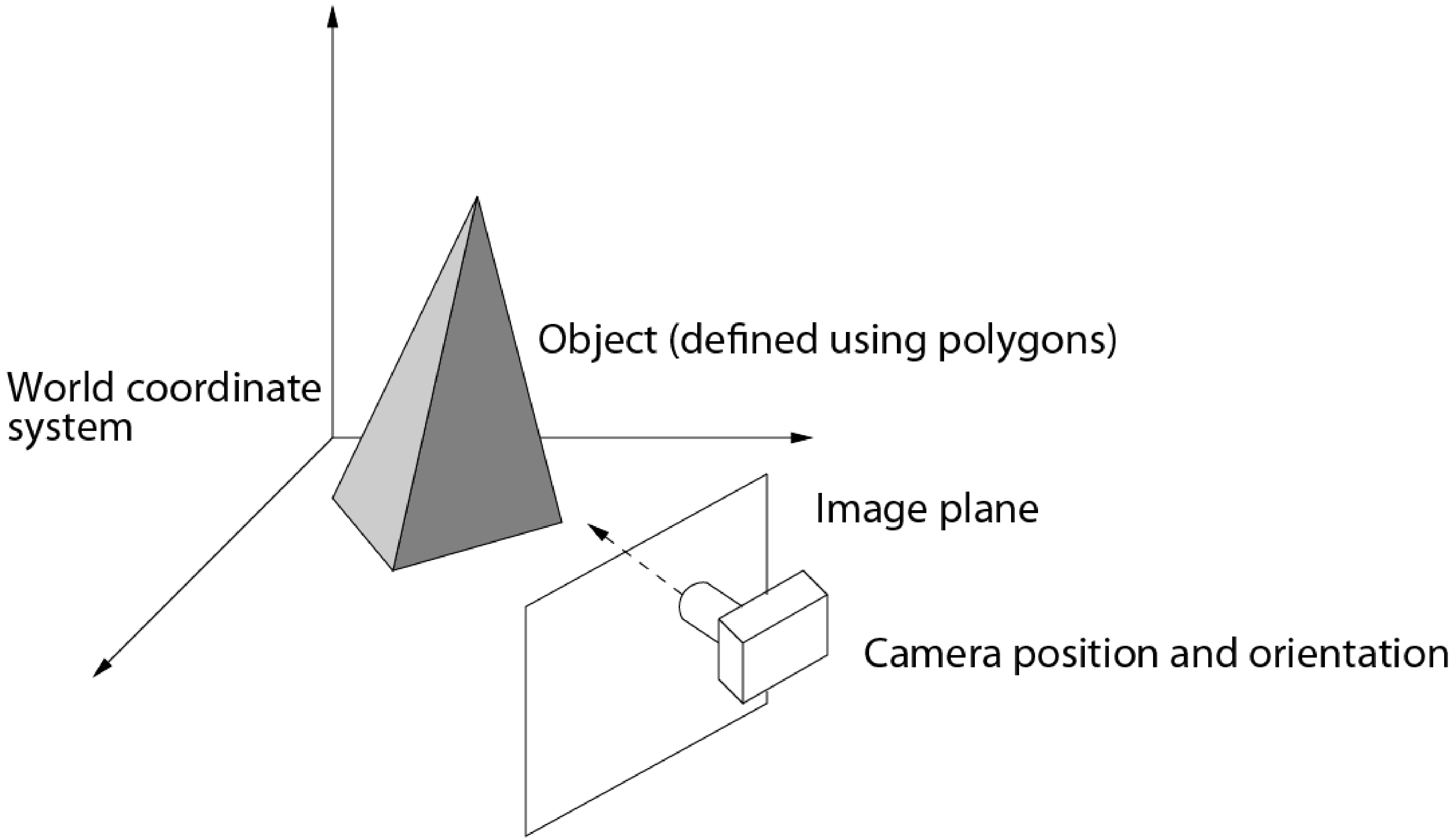
we know the solution is $t = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$

If the discriminant $b^2 - 4ac$ is negative, there is no solution (this corresponds to the ray missing the sphere).

If the discriminant is positive, there are two solutions and we should take the closest in front of the view point (smallest positive t)

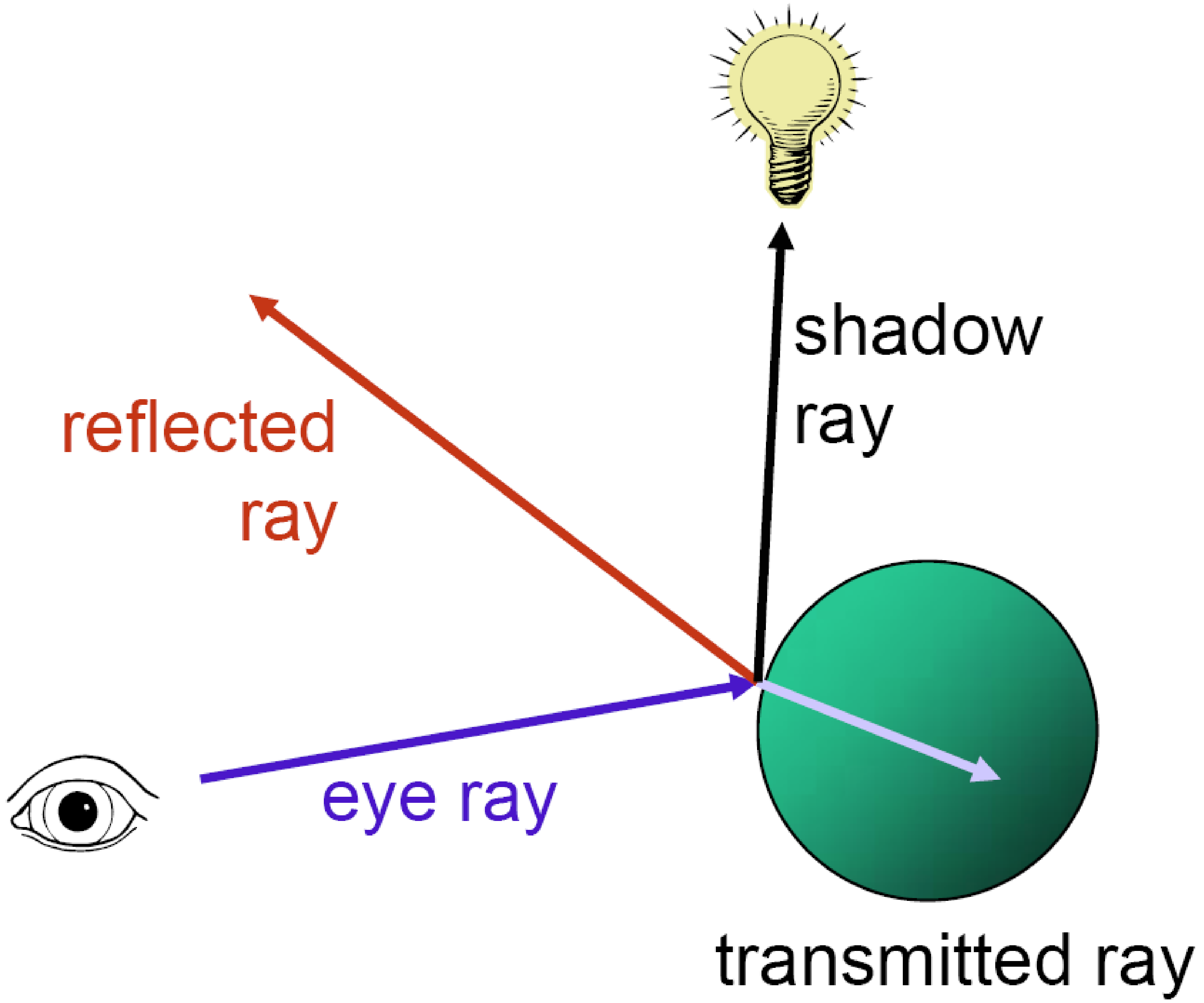
If the discriminant is zero, there is one solution (the ray was tangential to the sphere, hitting it at a single point)

World Coordinates



Code (for reference)

```
    procs=omp_get_num_procs();
    omp_set_num_threads(procs);
#pragma omp parallel private(tid, i, j, C, ray_orig, my_RayTri)
{
    tid=omp_get_thread_num();
    for (j = tid; j < tex_h; j+=procs) {
        for (i = 0; i < tex_w; i++) {
            /*** Calculate ray origin ***/
            ray_orig=my_camera.Ray(((double)(i-
centreX))/((double) tex_w), ((double)(j-centreY))/((double)
tex_h));
            my_RayTri.SetOrigin(ray_orig);
            my_RayTri.SetDir(my_camera.VPN);
            C=my_RayTri.TraceRay();
            *(*Image+i*4+j*tex_w*4)=(GLubyte) C.x;
            *(*Image+i*4+j*tex_w*4+1)=(GLubyte) C.y;
            *(*Image+i*4+j*tex_w*4+2)=(GLubyte) C.z;
            *(*Image+i*4+j*tex_w*4+3)= (GLubyte) 255;
        }
    }
}
```



shadow
ray
reflected
ray
eye ray
transmitted ray