COMP3421

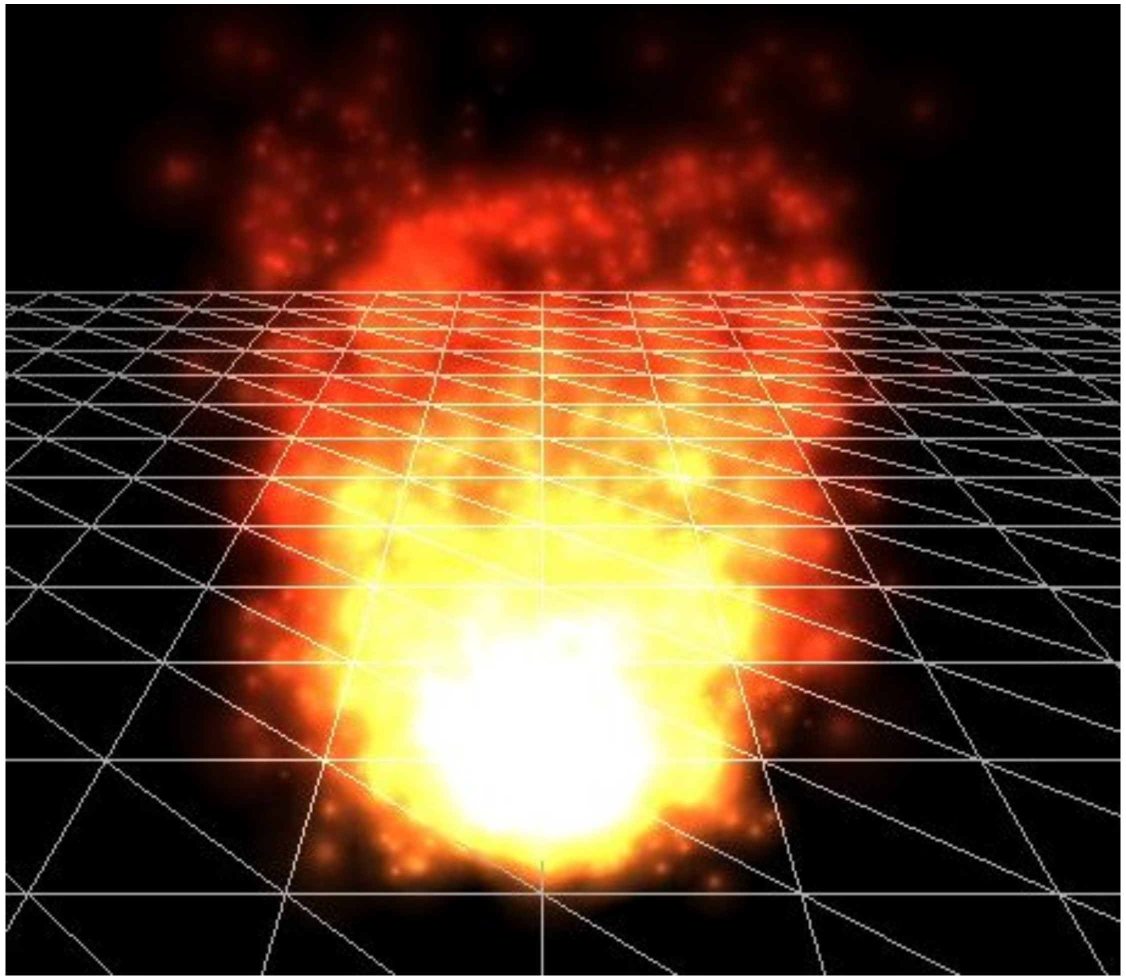
Particle Systems, Ray Tracing

Robert Clifton-Everest

Email: robertce@cse.unsw.edu.au

Particle systems

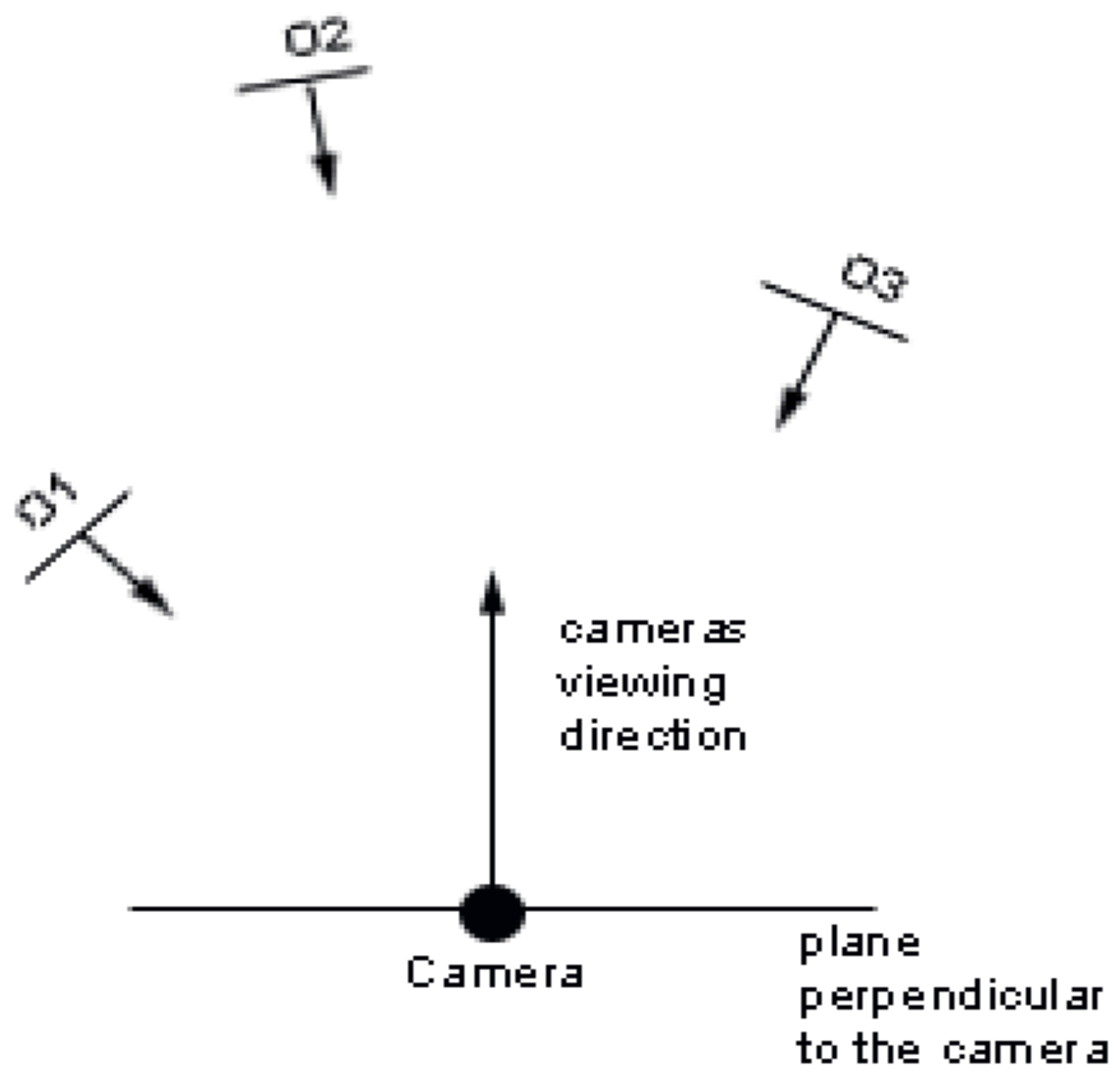
- Some visual phenomena are best modelled as collections of small particles.
- Examples: rain, snow, fire, smoke, dust



Particle systems

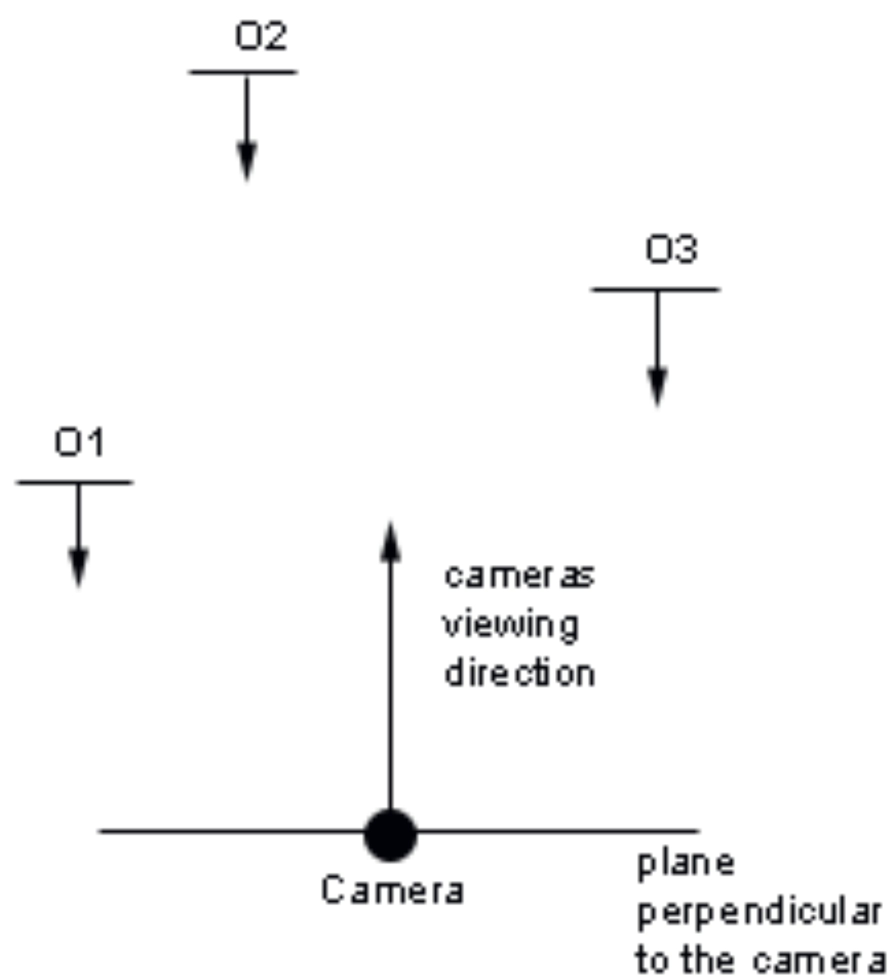
- Particles are usually represented as small textured quads or point sprites – single vertices with an image attached.
- They are billboarded, i.e transformed so that they are always face towards the camera.

Billboarding



Billboarding

- An approximate form of billboarding can be achieved by having polygons face a plane perpendicular to the camera



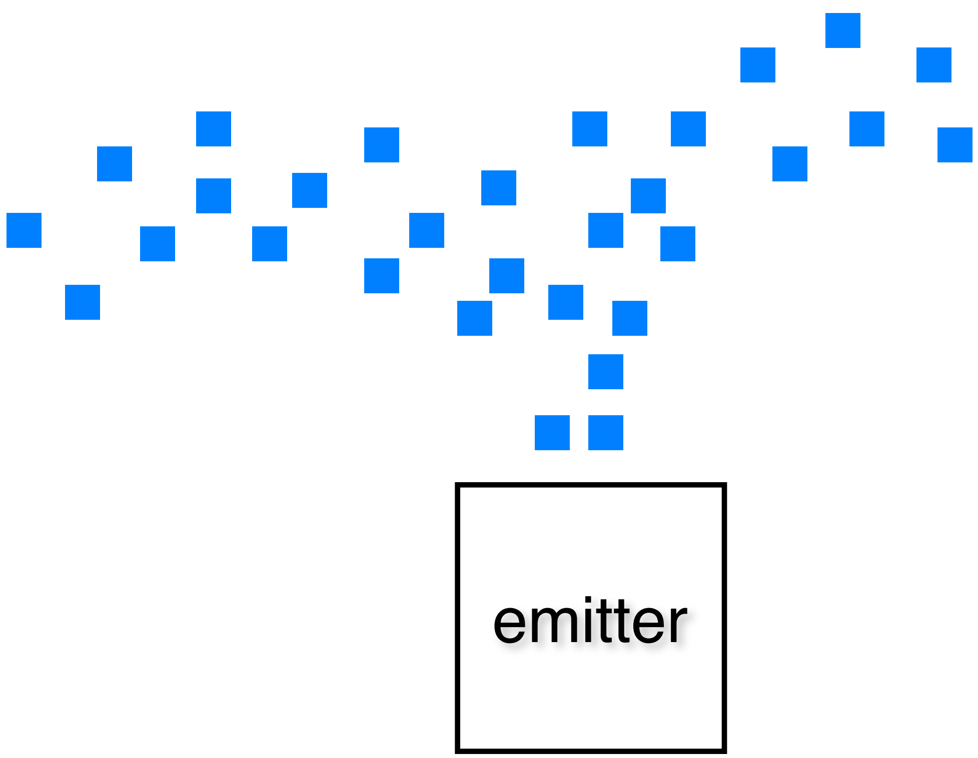
Point Sprites

- Using `GL_POINTS` to draw a textured sprite.
- Points have position, but no rotation or scale, so are implicitly billboarded
- Size of points can be set with

```
gl.glPointSize(int)
```

Particle systems

- Particles are created by an emitter object and evolve over time, usually changing position, size, colour.



Particle evolution

- Usually the rules for particle evolution are simple local equations:
  - interpolate from one colour to another over time
  - move with constant speed or acceleration.
- To simulate many particles it is important these update steps are kept simple and fast.

Particles on the GPU

- Particle systems are well suited to implementation as vertex shaders.
- The particles can be represented as individual vertices.
- A vertex shader can compute the position of each particle at each moment in time.

Exercise

- Adapt the fireworks example so that particle calculations are performed in the shader.

Global Lighting

- The lighting equation we looked at earlier only handled direct lighting from sources:

$$I = \boxed{I_a \rho_a} + \sum_{l \in \text{lights}} I_l \left(\rho_d (\hat{\mathbf{s}}_\mathbf{l} \cdot \hat{\mathbf{m}}) + \rho_{sp} \left(\hat{\mathbf{r}}_\mathbf{l} \cdot \hat{\mathbf{v}} \right)^f \right)$$

- We added an ambient fudge term to account for all other light in the scene.
- Without this term, surfaces not facing a light source are black.

Story so far…

Global lighting

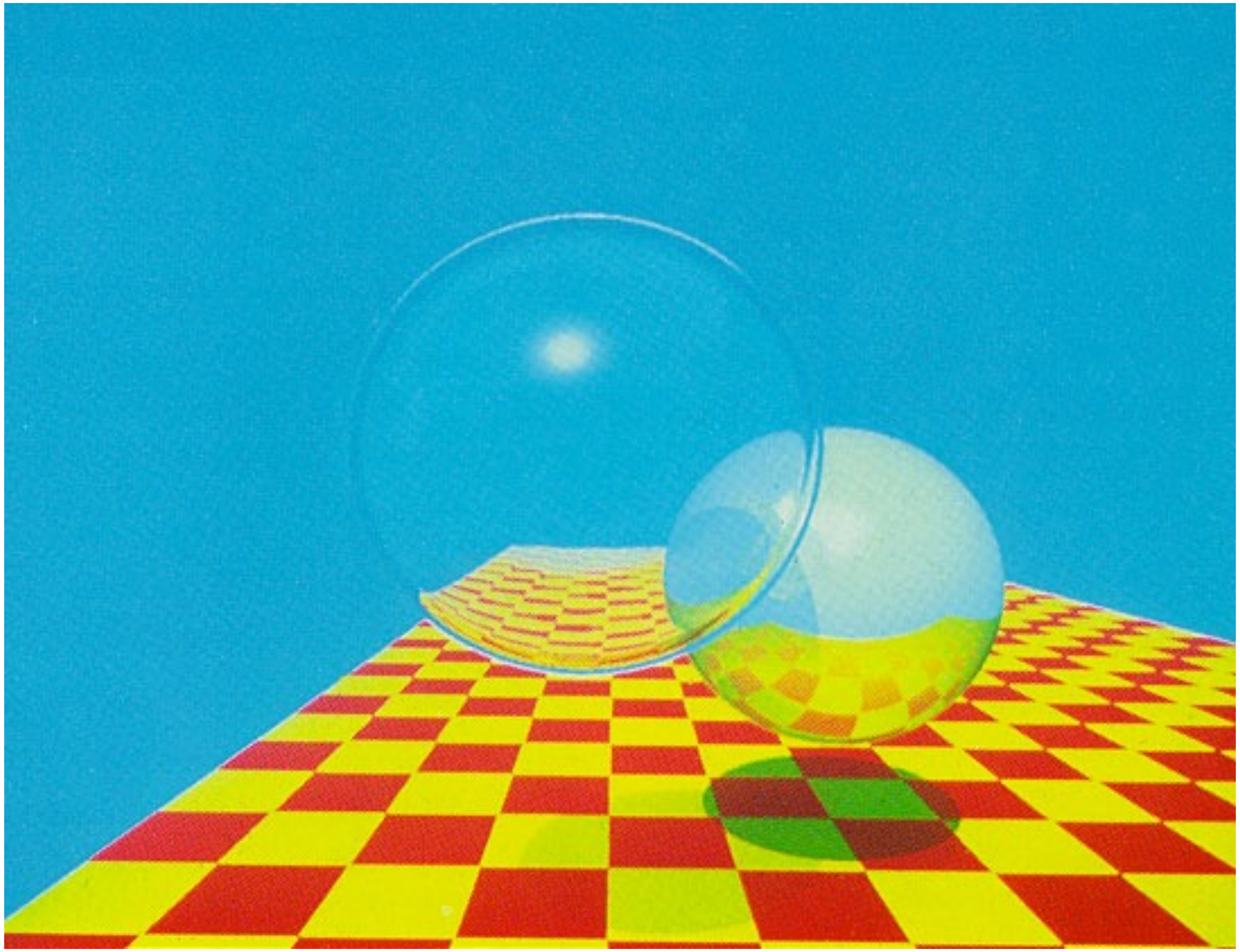
- In reality, the light falling on a surface comes from everywhere. Light from one surface is reflected onto another surface and then another, and another, and...

- Methods that take this kind of multi-bounce lighting into account are called global lighting methods.

Raytracing and Radiosity

- There are two main methods for global lighting:
  - Raytracing models specular reflection and refraction.
  - Radiosity models diffuse reflection.
- Both methods are computationally expensive and are rarely suitable for real-time rendering.

Ray Tracing – 1980s

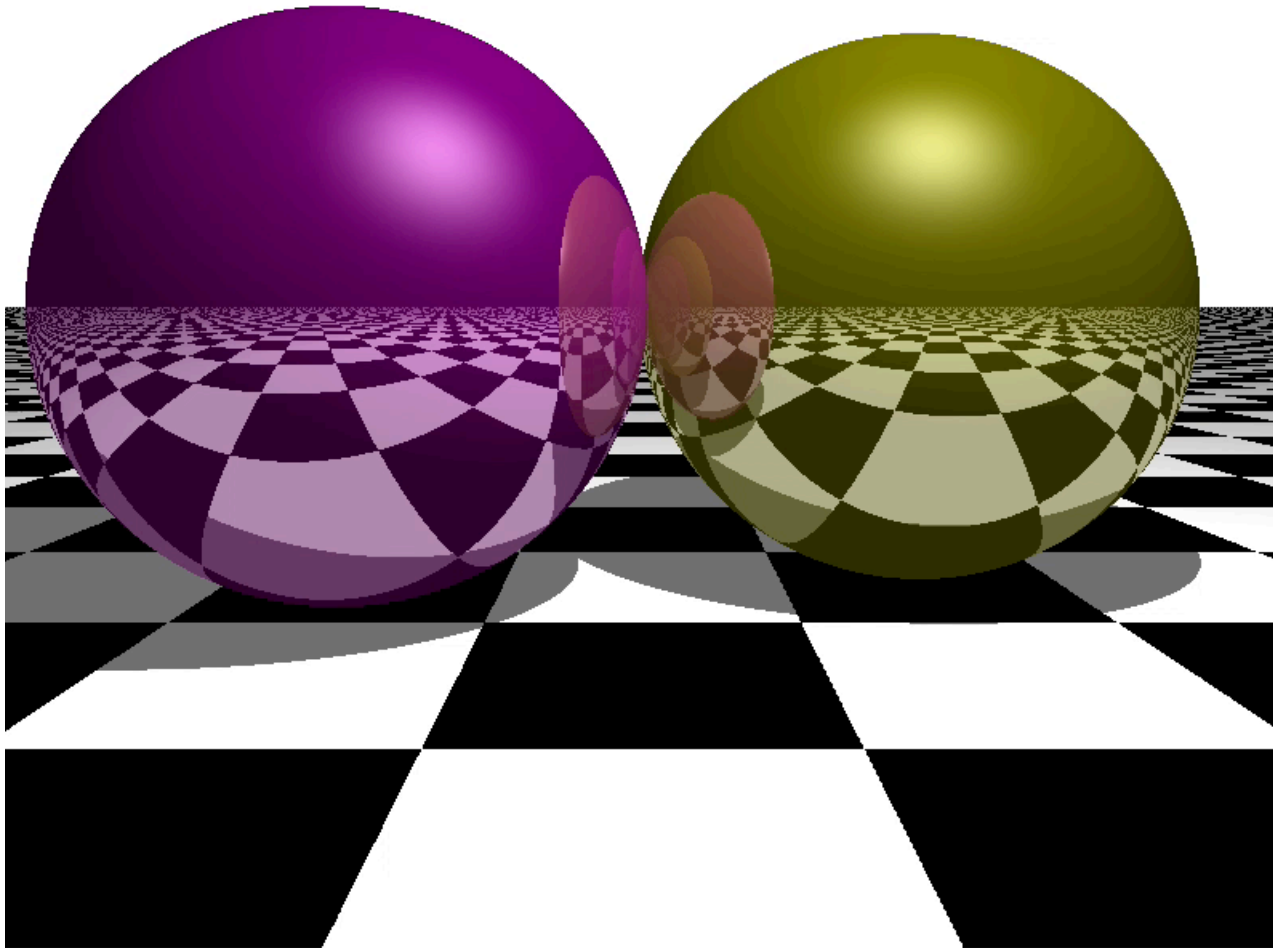


Ray tracing - 2006

Ray tracing - 2018

- https://www.youtube.com/watch?v=J3ue35ago3Y

Ray Tracing - COMP3421



Ray tracing

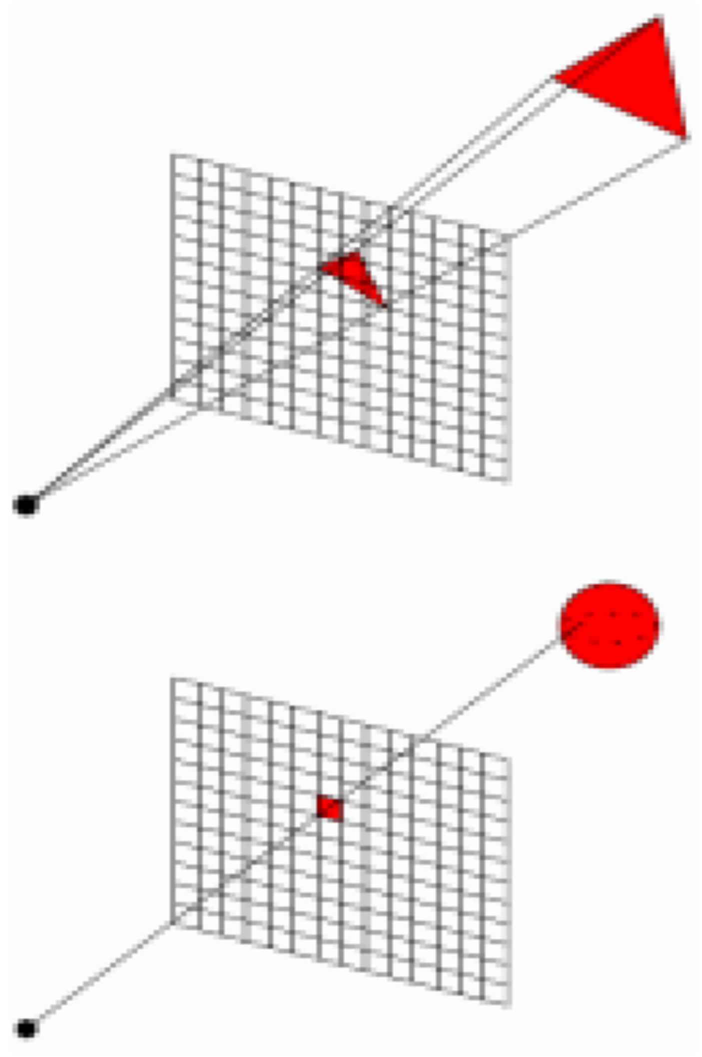
- Ray tracing is a different approach to rendering than the pipeline we have seen so far.

- In the OpenGL pipeline we model objects as meshes of polygons which we convert into fragments and then display (or not).

- In ray tracing, we model objects as implicit forms and compute each pixel by casting a ray and seeing which models it intersects.

Projective Methods vs Ray Tracing

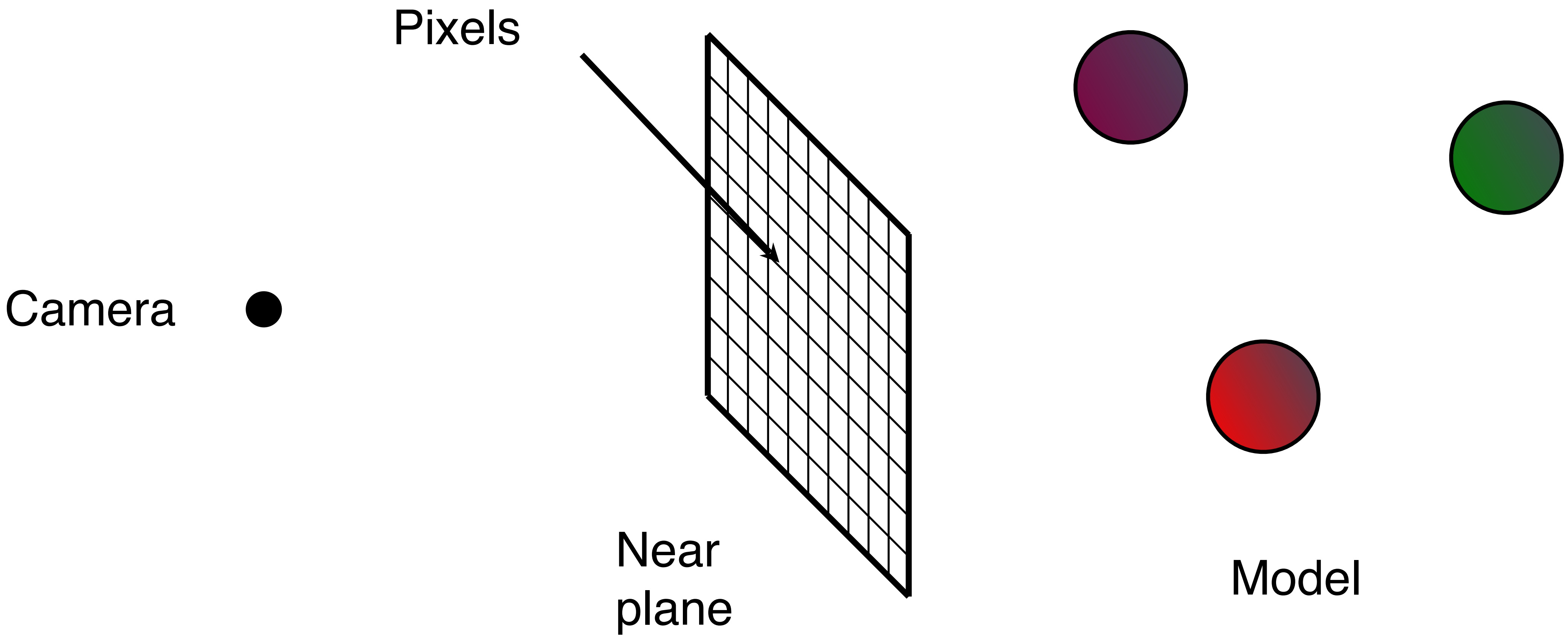
- Projective Methods:
  - For each **object:** Find and update each pixel it influences
- Ray Tracing:
  - For each **pixel:** Find each object that influences it and update accordingly



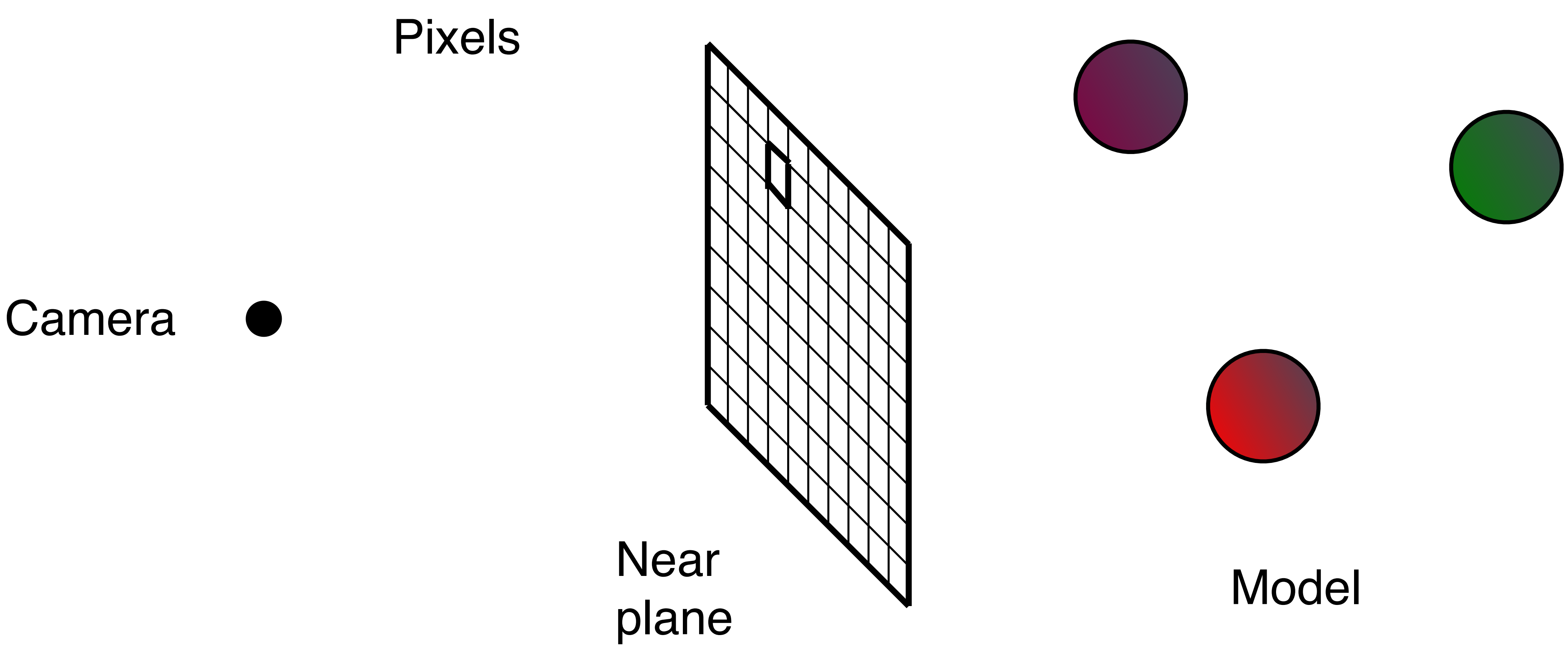
Projective Methods vs Ray Tracing

- They share lots of techniques:
  - shading models,
  - calculation of intersections,
- They also have differences:
  - projection and hidden surface removal come for ‘free’ in ray tracing

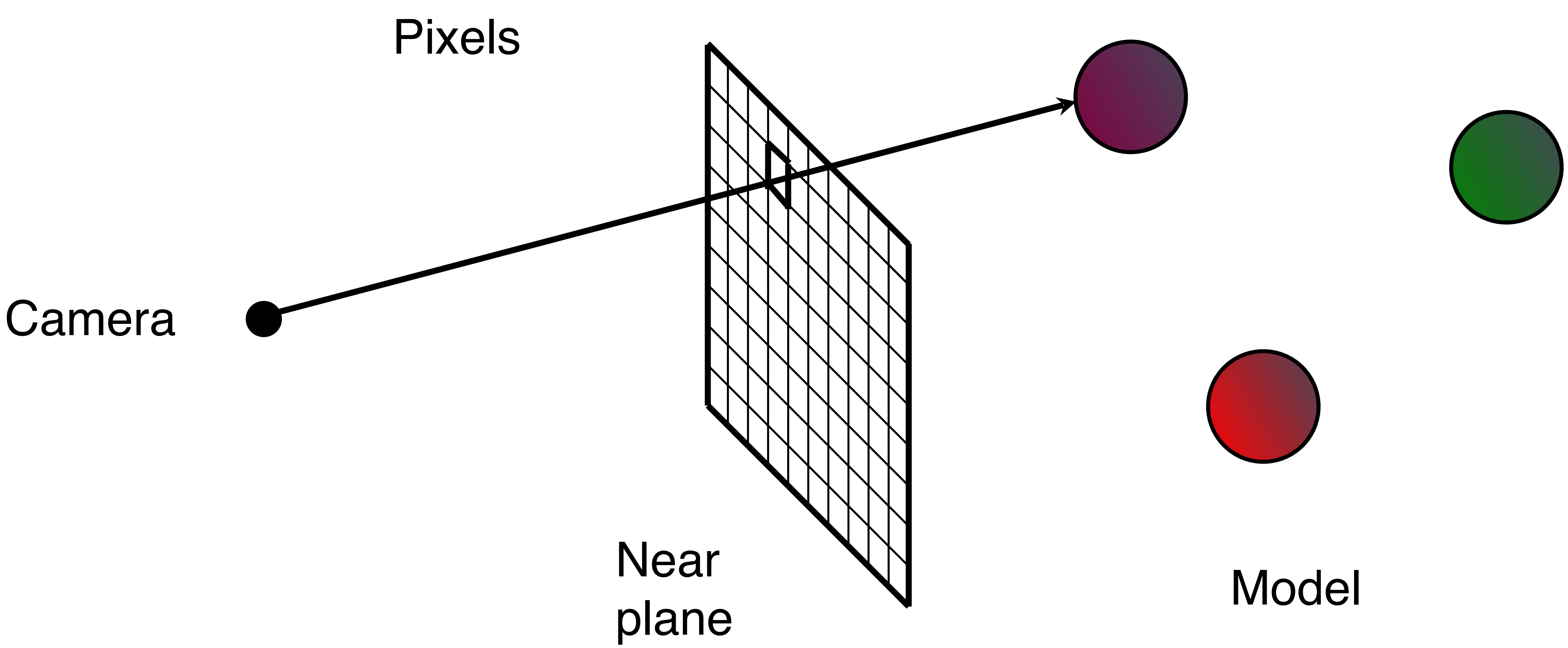
Rays



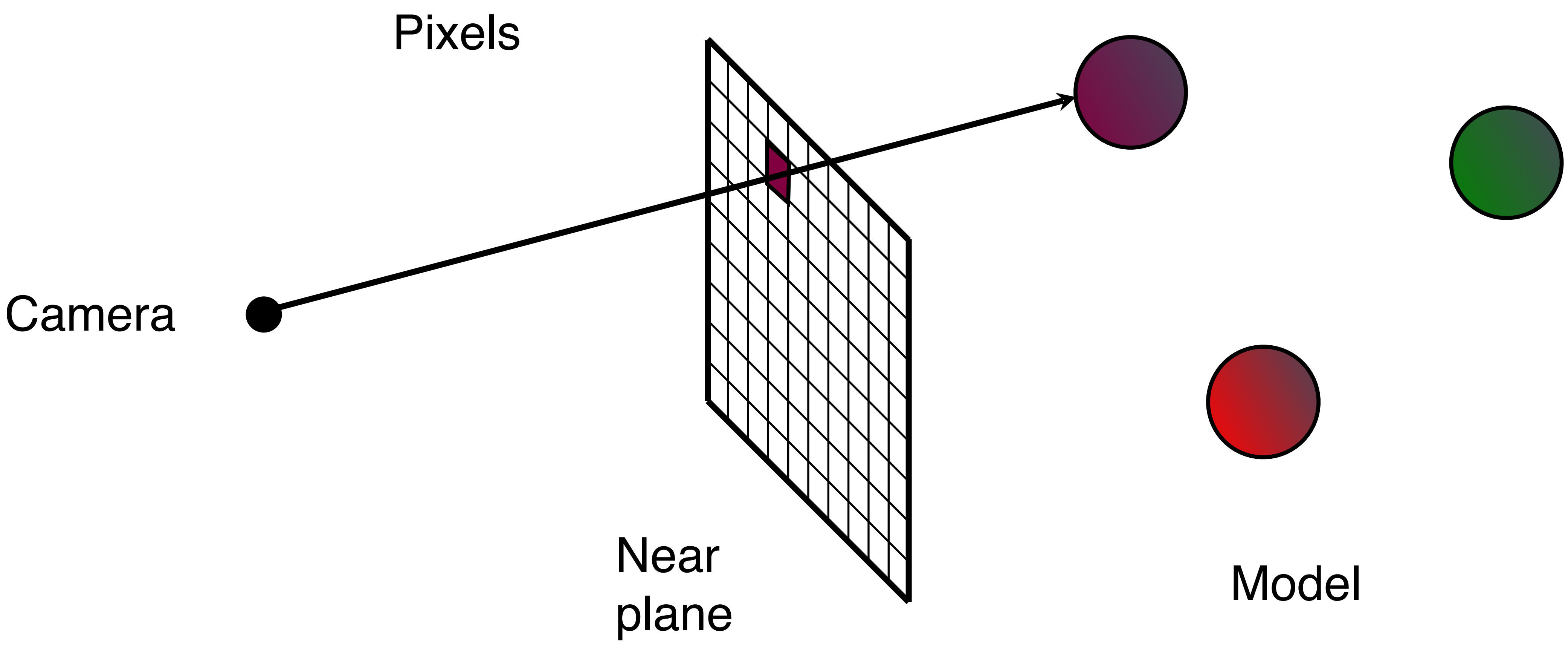
Rays



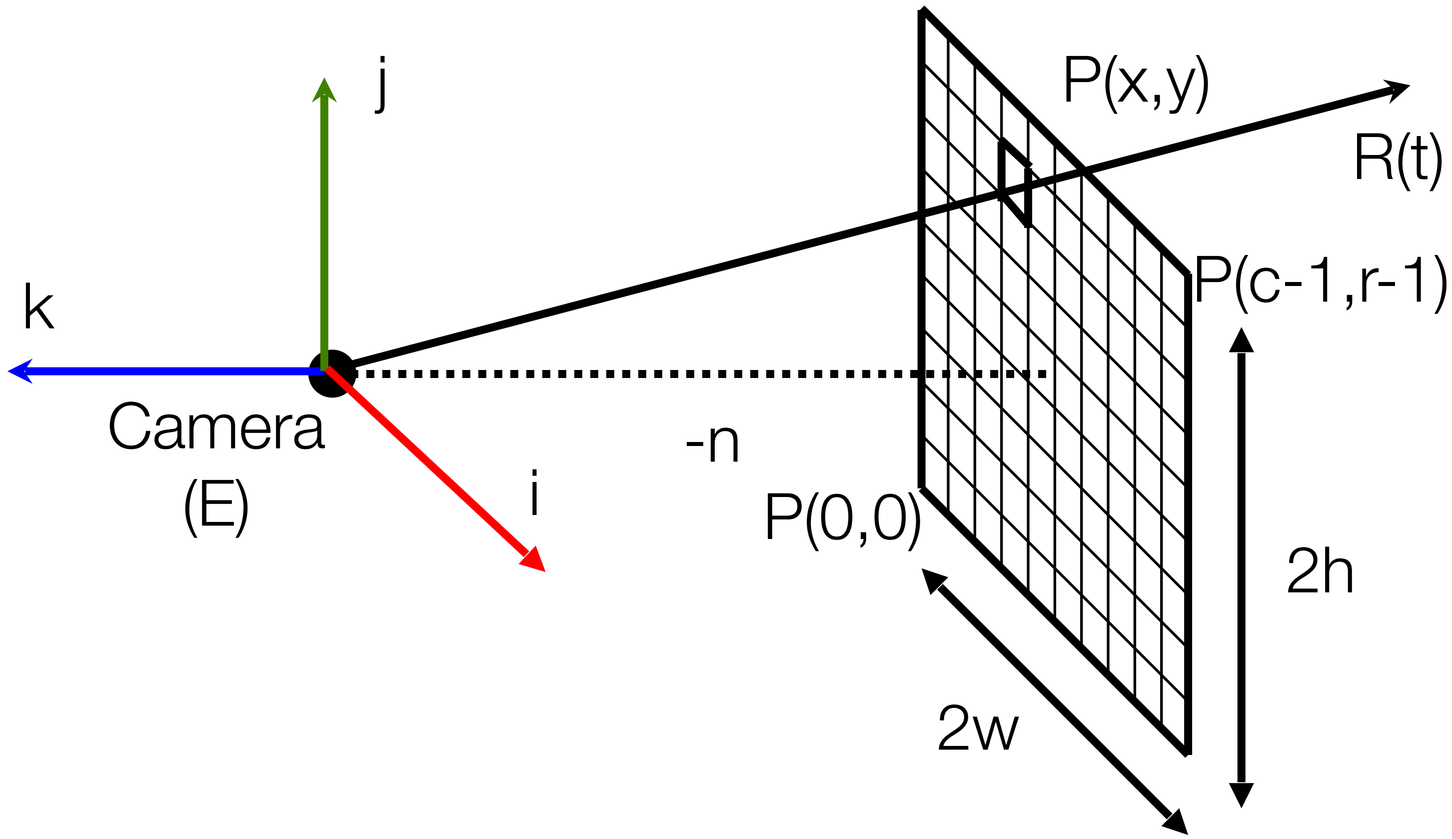
Rays



Rays



Rays



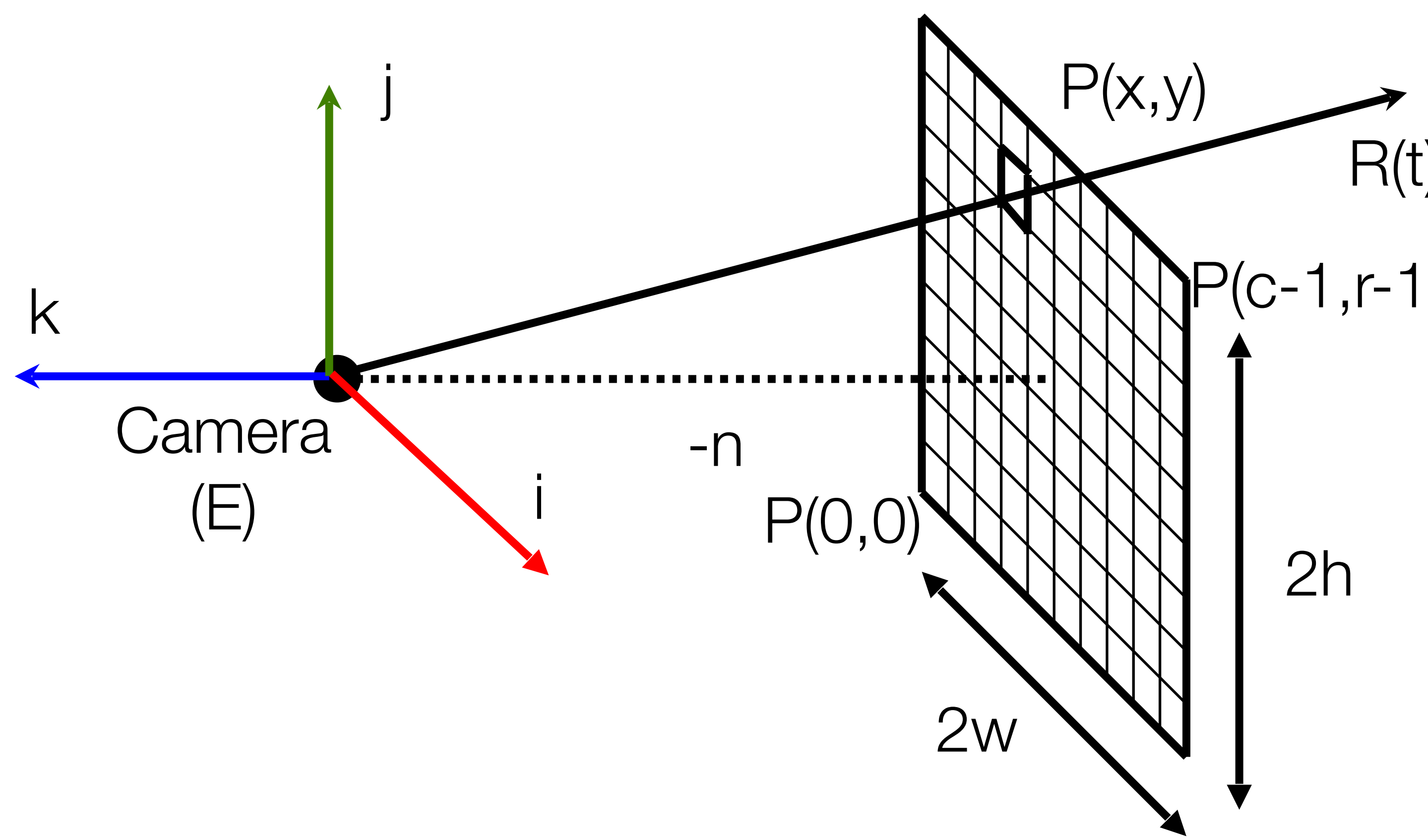
Location of Pixels

- Where on the near plane does a given pixel (x,y) appear? (Lower left corners of pixels)

$$pixelWidth = \frac{2w}{c}$$

$$i_c = -w + x\left(\frac{2w}{c}\right)$$

$$= w\left(\frac{2x}{c} - 1\right)$$

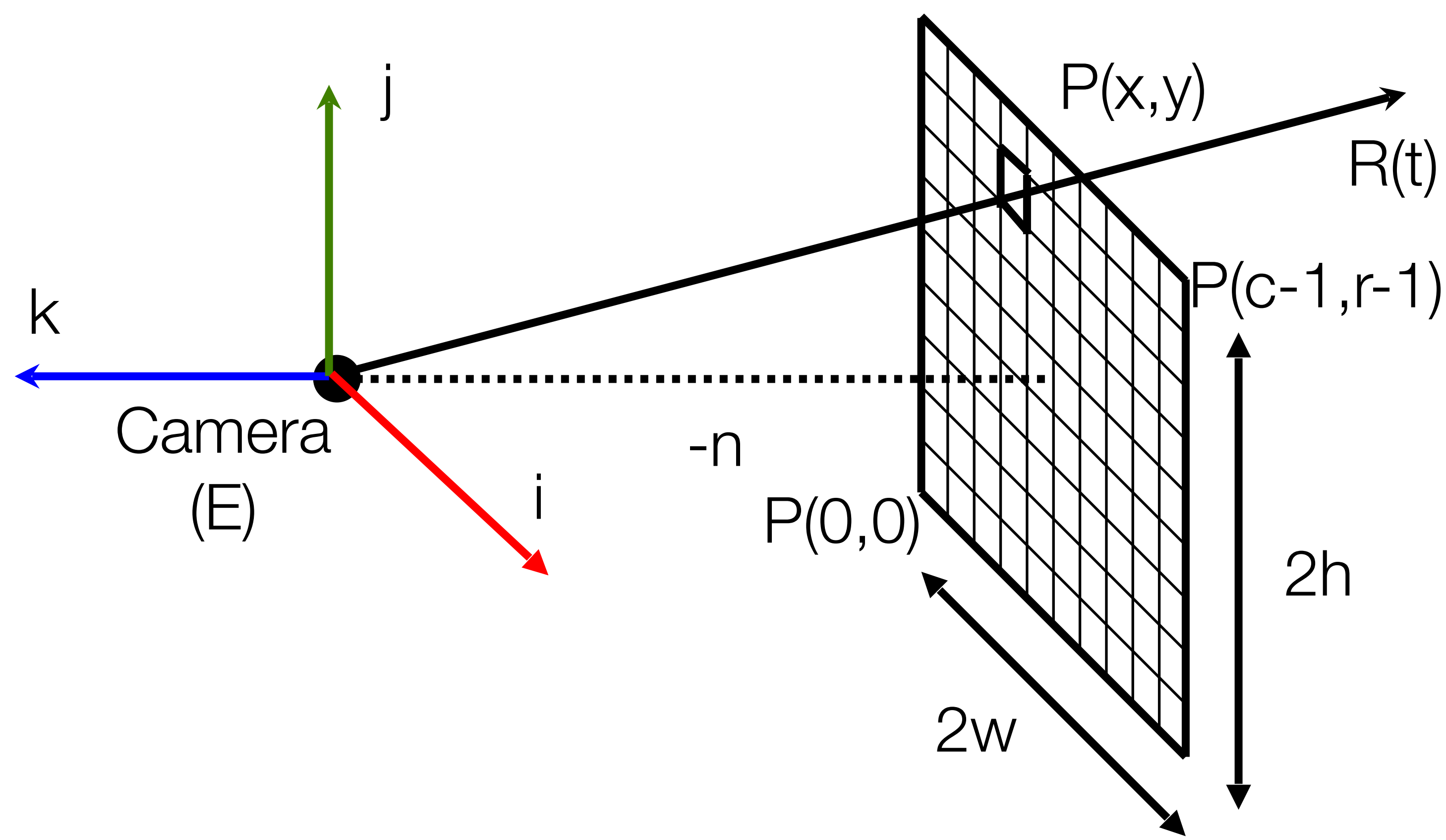


Location of Pixels

- Where on the near plane does a given pixel (x,y) appear? (Lower left corners of pixels)

$$pixelHeight = \frac{2h}{r}$$

$$j_r = h\left(\frac{2y}{r} - 1\right)$$



Rays

- The point P(x,y) of pixel (x,y) is given by:

$$P(x,y) \quad = \quad E + w(\tfrac{2x}{c} - 1)\mathbf{i} + h(\tfrac{2y}{r} - 1)\mathbf{j} - n\mathbf{k}$$

- A ray from the camera through P(x,y) is given by:

$$\begin{aligned} R(t) &= E + t(P(x,y) - E) \\ &= E + t\mathbf{v} \\ \mathbf{v} &= w(\tfrac{2x}{c} - 1)\mathbf{i} + h(\tfrac{2y}{r} - 1)\mathbf{j} - n\mathbf{k} \end{aligned}$$

Rays

$$R(t) = E + t(P(x,y) - E)$$

$$= E + t\mathbf{v}$$

When:

$t = 0$, we get E (Eye/Camera)

$t = 1$, we get P(x,y) – the point on the near plane

$t > 1$ point in the world

$t < 0$ point behind the camera – not on ray

Intersections

- We want to compute where this ray intersects with objects in the scene.
- For basic shapes, we can do this with the equation of the shape in implicit form:

$$F(x, y, z) = 0$$

- which we can also write as:

$$F(P) = 0$$

- We substitute the formula for the ray into F and solve for t.

Intersecting a generic sphere

- For example, a unit sphere at the origin has implicit form:

$$F(x, y, z) = x^2 + y^2 + z^2 - 1 = 0$$

or:

$$F(P) = |P|^2 - 1 = 0$$

Intersecting a generic sphere

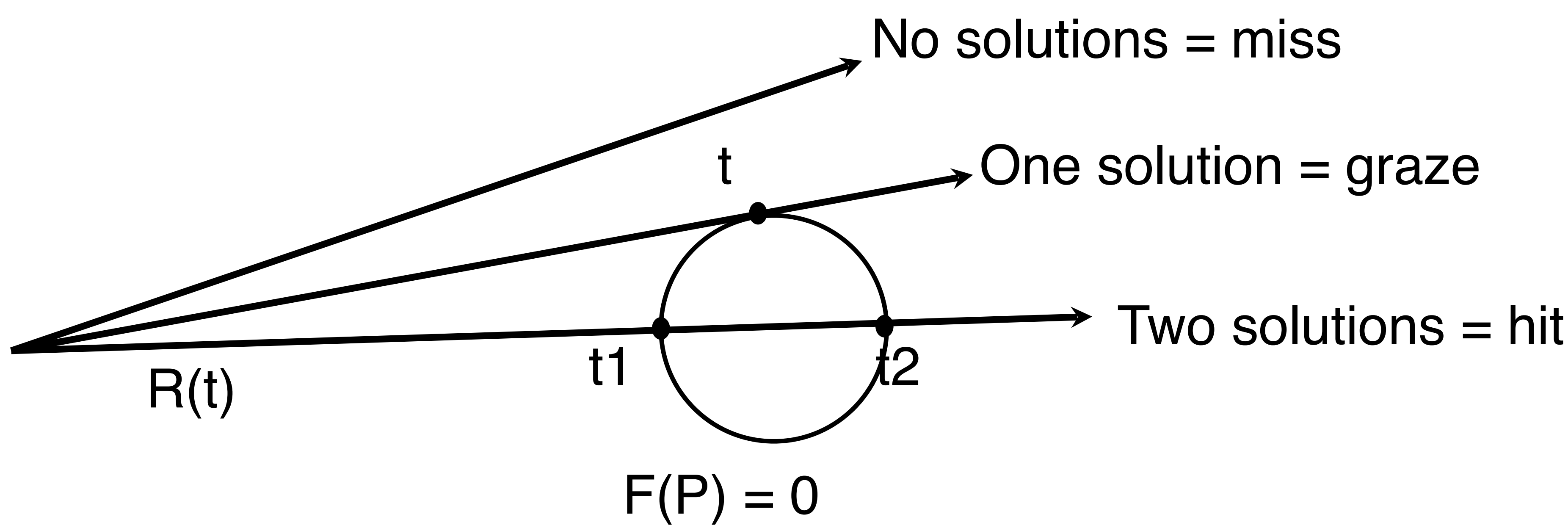
- We substitute the ray equation into F and solve for t:

$$\begin{aligned} F(R(t)) &= 0 \\ |R(t)|^2 - 1 &= 0 \\ |E + \mathbf{v}t|^2 - 1 &= 0 \\ |\mathbf{v}|^2 t^2 + 2(E \cdot \mathbf{v})t + (|E|^2 - 1) &= 0 \end{aligned}$$

- which we can solve for t (as a quadratic).

Intersecting a generic sphere

- We will get zero, one or two solutions:



$$t_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

Exercise

- Where is the intersection of

$$R(t) = (3,2,3) + (-3,-2,-3)t$$

- With the generic sphere?

Intersecting a generic plane

- The x-y plane has implicit form:

$$F(x,y,z) = z = 0$$
$$F(P) = p_z = 0$$

- Intersecting with the ray:

$$\begin{aligned} F(R(t)) &= 0 \\ E_z + t\mathbf{v}_z &= 0 \\ t &= -\frac{E_z}{\mathbf{v}_z} \end{aligned}$$

Intersecting a generic cube

- To compute intersections with the generic cube (-1,-1,-1) to (1,1,1) we apply the Cyrus-Beck clipping algorithm encountered in week 3. Extending the algorithm to 3D is straightforward.

- The same algorithm can be used to compute intersections with arbitrary convex polyhedral and meshes of convex faces.

Non-generic solids

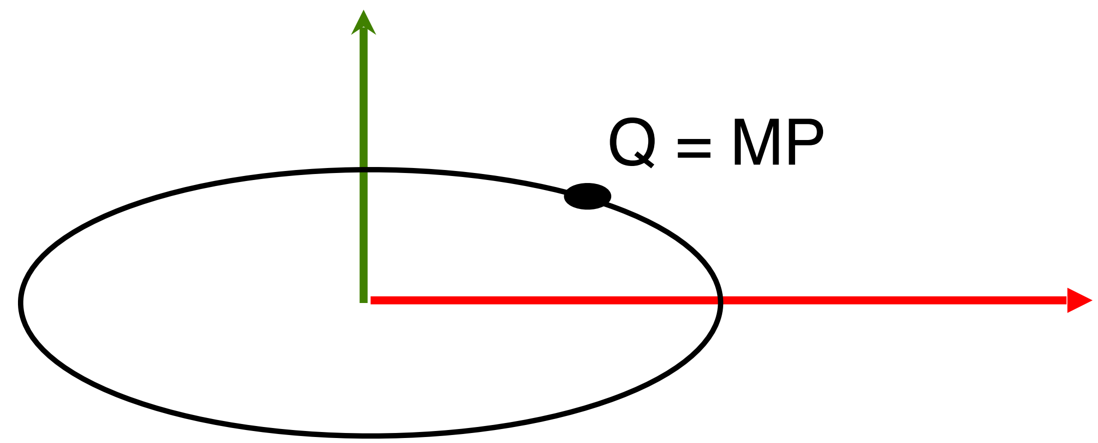
- We can avoid writing special-purpose code to calculate intersections with non-generic spheres, boxes, planes, etc.

- Instead we can transform the ray and test it against the generic version of the shape.

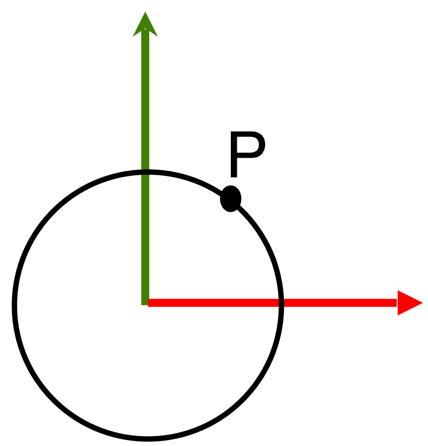
Transformed spheres

- We can transform a sphere by applying affine transformations
- Let P be a point on the generic sphere.
- We can create an arbitrary ellipsoid by transforming P to a new coordinate frame given by a matrix M.

2D example



Transformed circle
$F(M^{-1}Q) = 0$



Generic circle
$F(P) = 0$

$$
\begin{aligned}
F(P) &= 0 \\
Q &= \mathbf{M}P \\
P &= \mathbf{M}^{-1}Q \\
F(\mathbf{M}^{-1}Q) &= 0
\end{aligned}
$$

Non-generic solids

- So in general if we apply a coordinate transformation M to a generic solid with implicit equation F(P) = 0 we get:

$$
\begin{aligned}
F(\mathbf{M}^{-1}Q) &= 0 \\
F(\mathbf{M}^{-1}R(t)) &= 0 \\
F(\mathbf{M}^{-1}E + t\mathbf{M}^{-1}\mathbf{v}) &= 0
\end{aligned}
$$

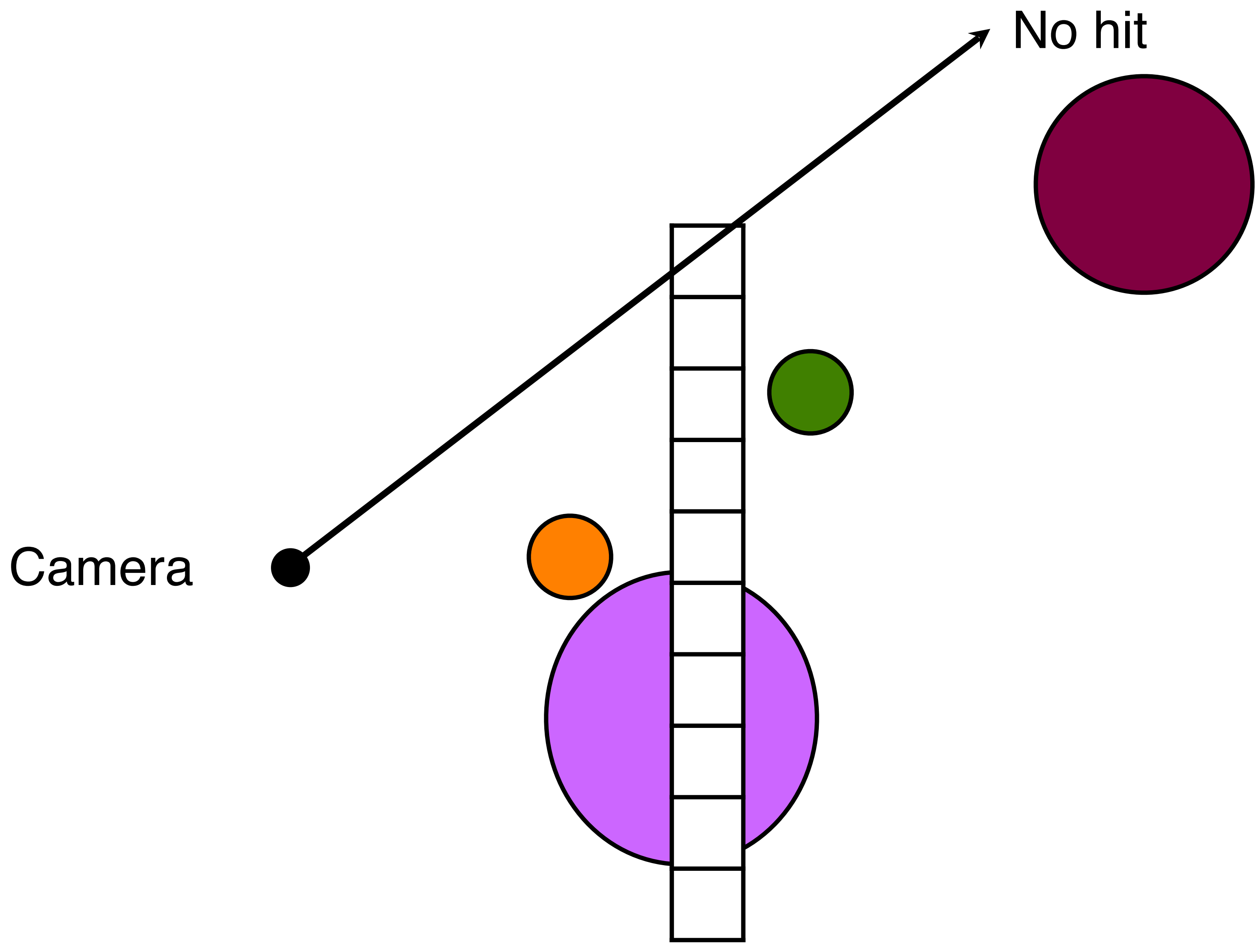
Non-generic Solids

- In other words:
  - Apply the inverse transformation to the ray.
  - Do standard intersection with the generic form of the object.
  - Affine transformations preserve relative distances so values of t will be valid.

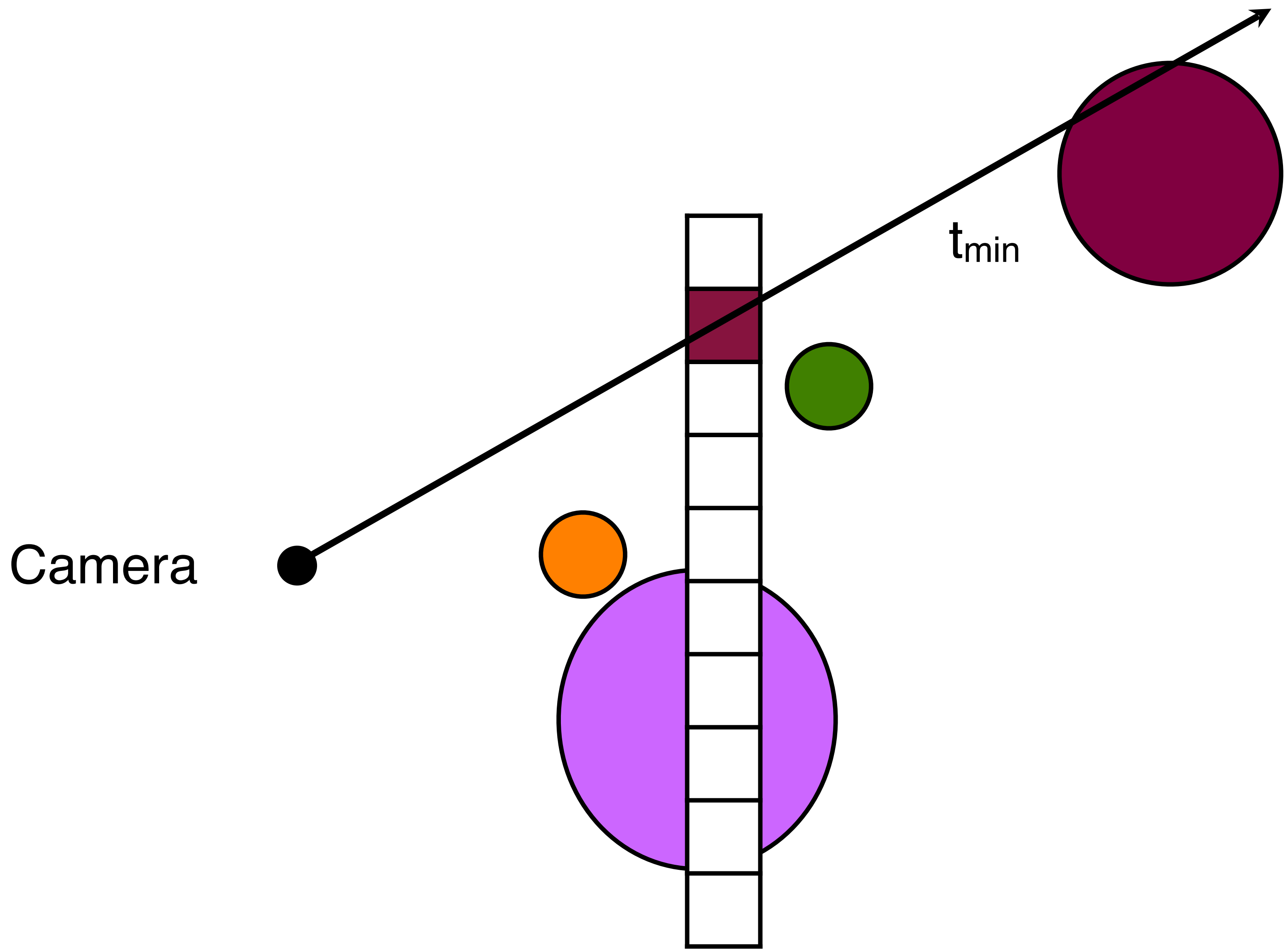
Ray Tracing Pseudocode

```
for each pixel (x,y):
    v = P(x,y) - E
    hits = {};
    for each object obj in the scene:
        E' = M-1 * E
        v' = M-1 * v
        hits.add(obj.hit(E', v'))
    hit = h in hits with min time > 1
    if (hit is null)
        set (x,y) to background
    else
        set (x,y) to hit.obj.colour(R(hit.time))
```

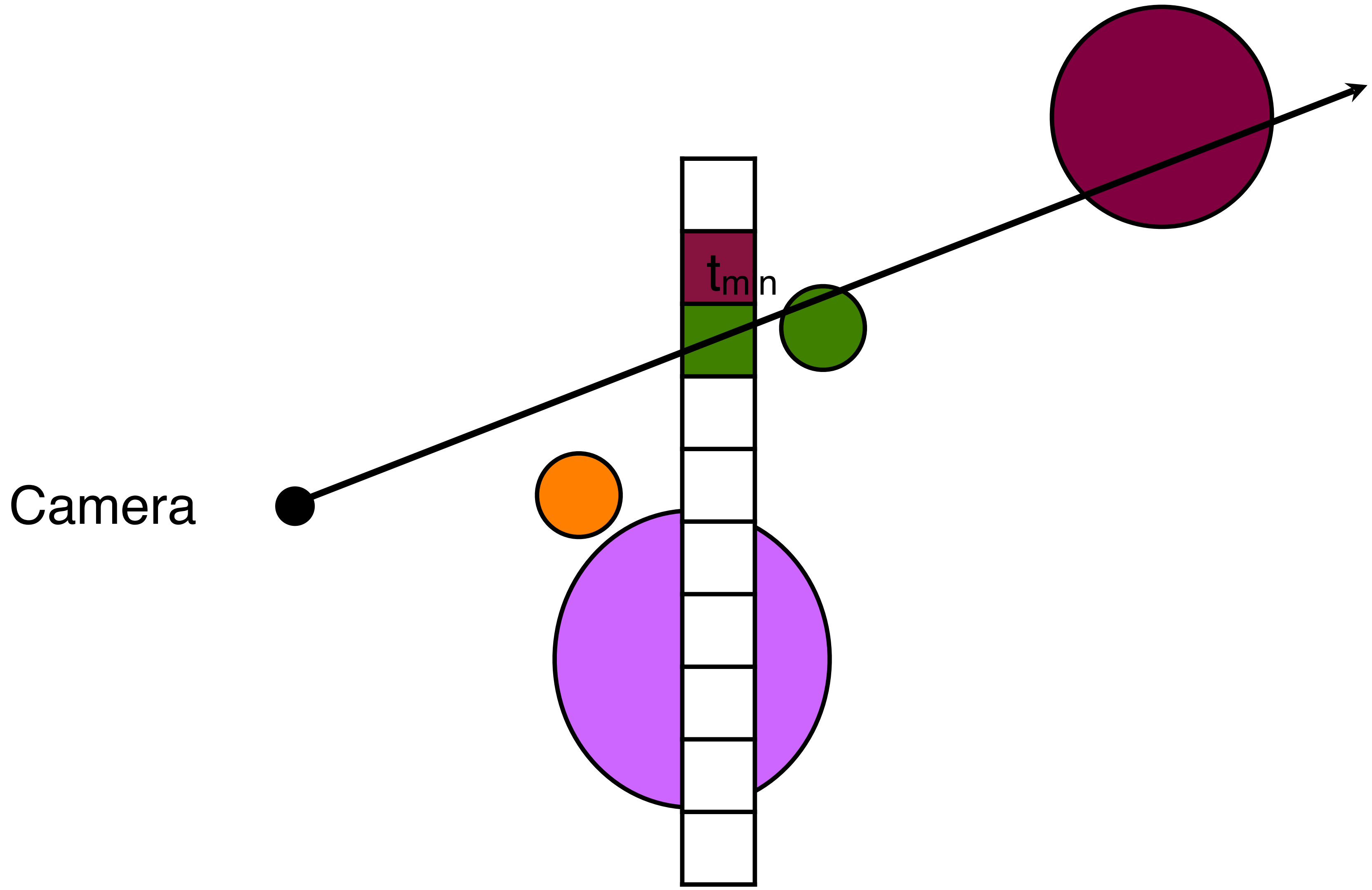
2D Example



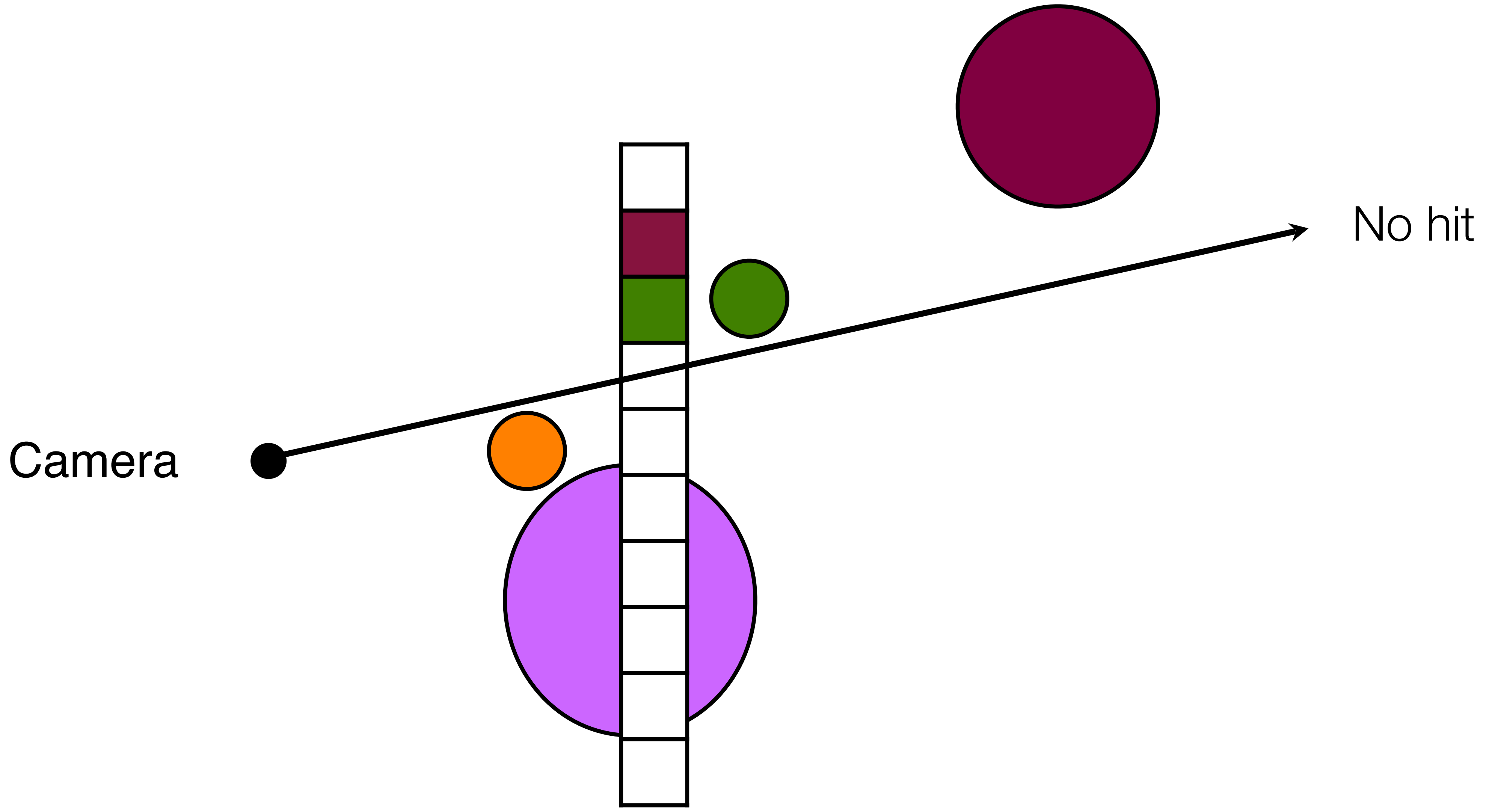
2D Example



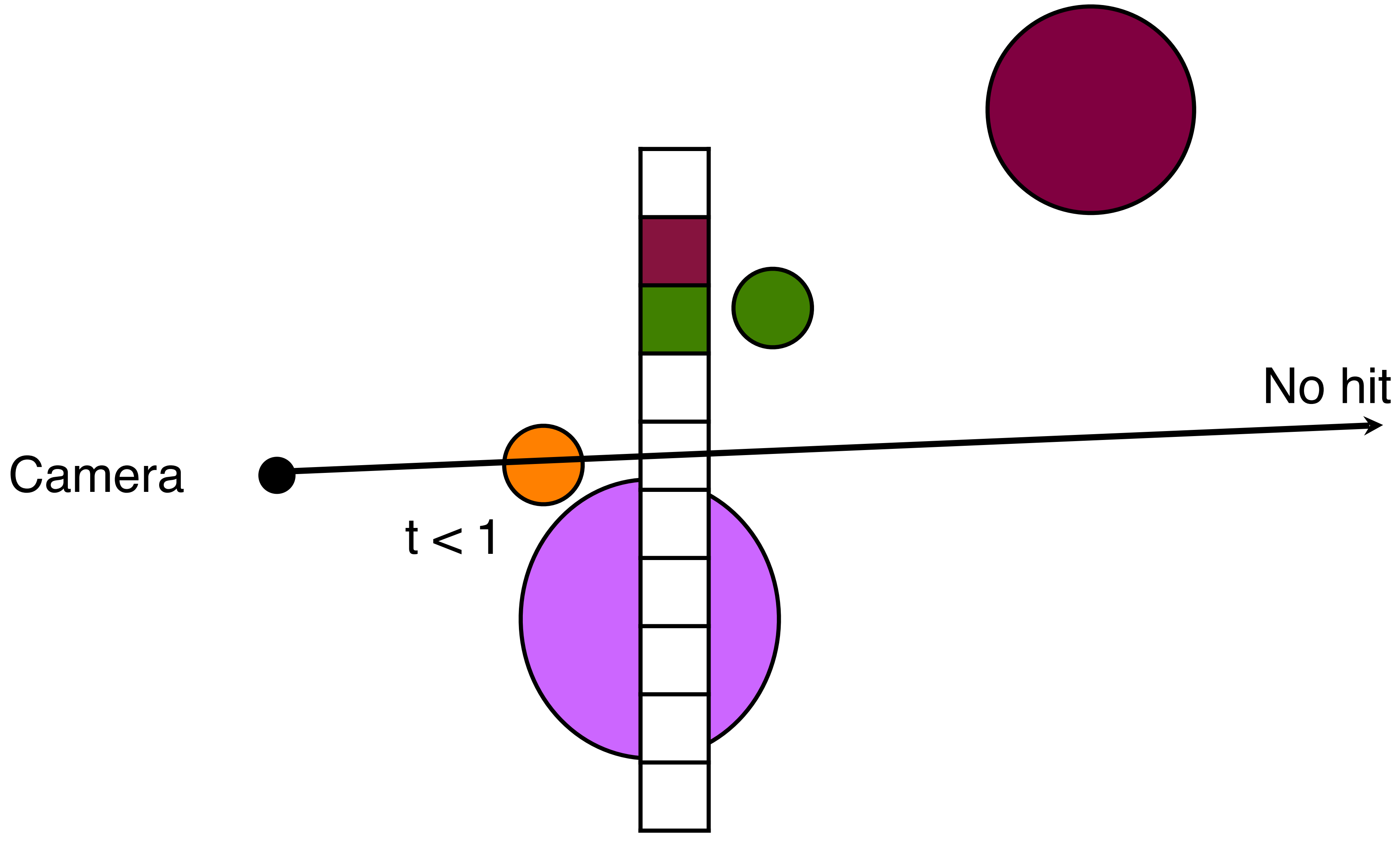
2D Example



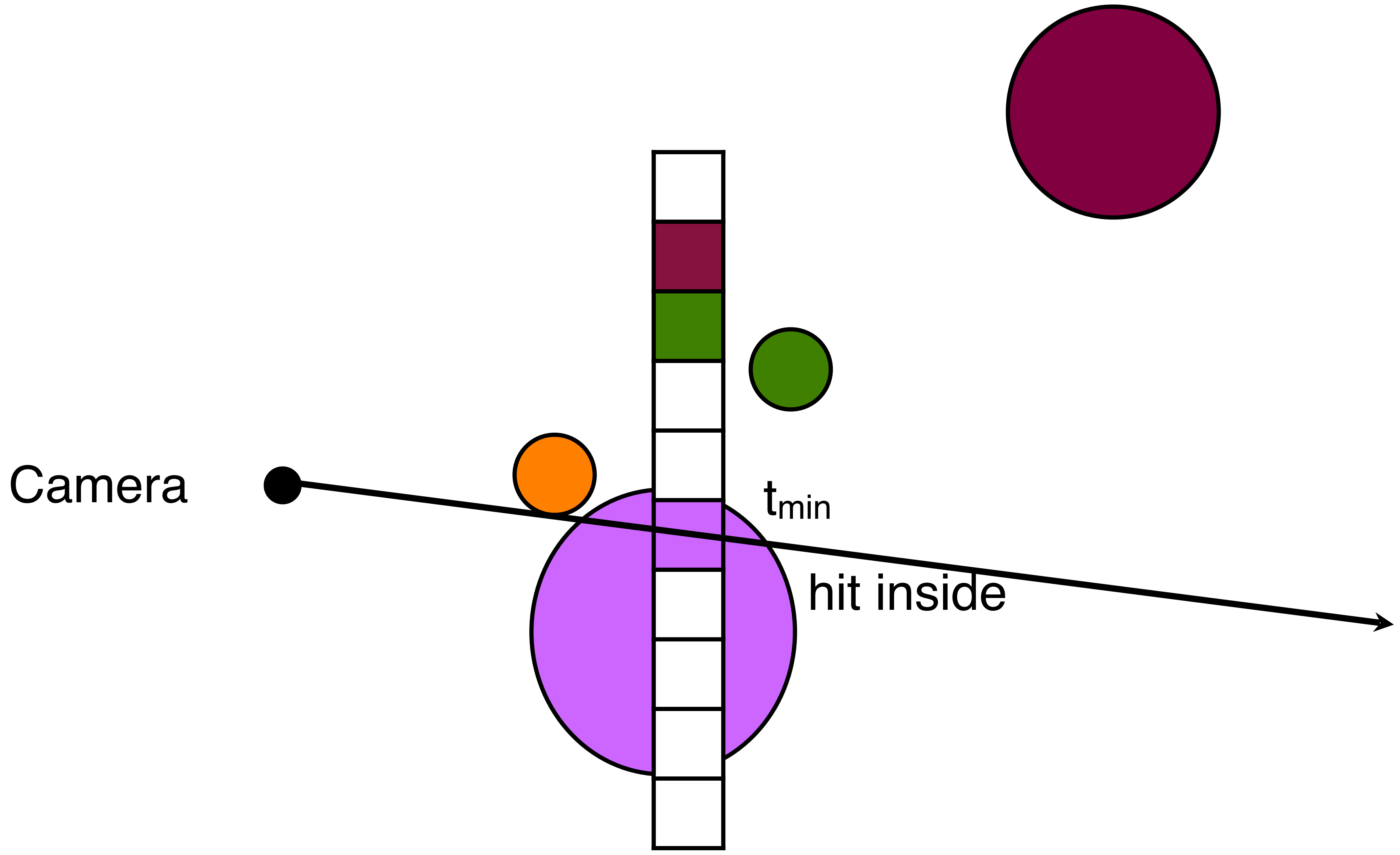
2D Example



2D Example



2D Example



Shading & Texturing

- When we know the object we hit and the point at which the hit occurs, we can compute the lighting equation to get the illumination.
- Likewise if the object has a texture we can compute the texture coordinates for the hit point to calculate its colour.
- We combine these as usual to compute the pixel colour.