CS4620/5620: Lecture 33

Animation and Ray Tracing

Announcements

- Quaternion problem, 3.3: 180 degrees
- 4621
    - Friday (animation): Nov 16

- Plan
    - Ray Tracing
        - Thanksgiving
    - Color
    - Prelim (Thu after Thanksgiving)

Physically-Based Motion

- Try to explicitly model the physics of motion

- Animate: human, birds
- Inanimate: fire, smoke, water, cloth

- Pro: captures reality
- Con: hard to control

Physically-Based Animation

- Must obey laws of physics

- Lot harder to simulate
  - Not just interpolation
  - Must solve for equilibrium solutions
    - Newtonian physics, Navier Stokes equations

Resources

- Physically Based Modeling Notes
  - http://www.pixar.com/companyinfo/research/pbm2001/index.html
    - Differential Equation Basics
    - Particle Dynamics
    - Rigid Body Dynamics

Overview

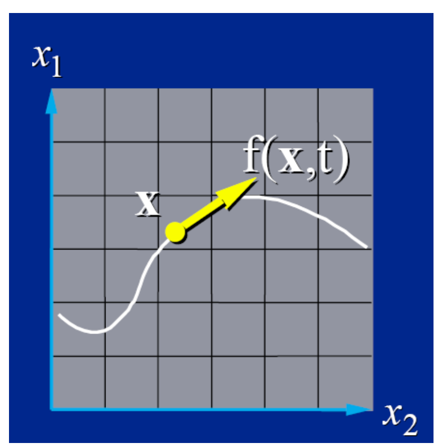
- Model with physical attributes
  - Mass, moment of inertia, elasticity, etc.
- Derive differential equations by applying Newtonian physics
- Specify initial conditions: position, velocity
- Specify external forces (maybe keyframe)
- Solve for motion

Ordinary Differential Equation (ODE)

- Have function f for derivative of x

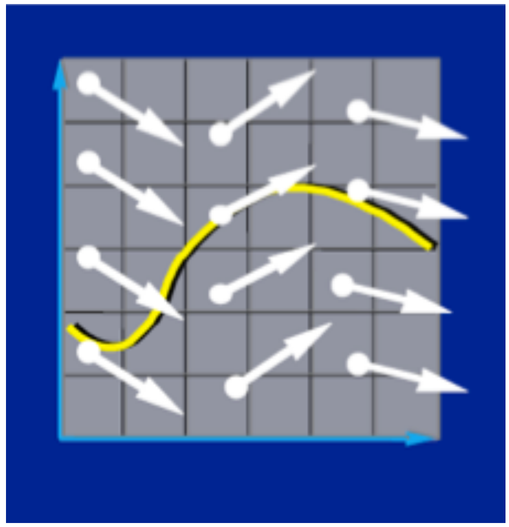
$$\dot{x} = f(x(t))$$

- x is state
  - x is a moving point
- f is known
  - f is its velocity



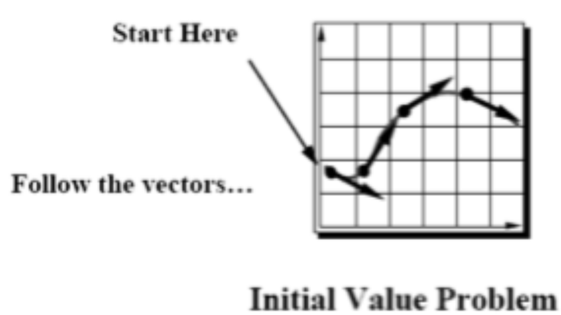
Vector Field

- The differential equation defines a vector field over x

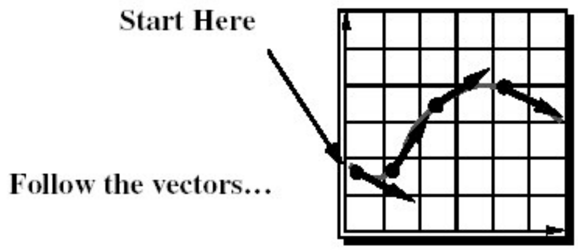


Initial Value Problem

- We have an initial value for x: x(t0)
- We want to solve for x over time
- How do we do it?
  – Numerical solution



Euler Method

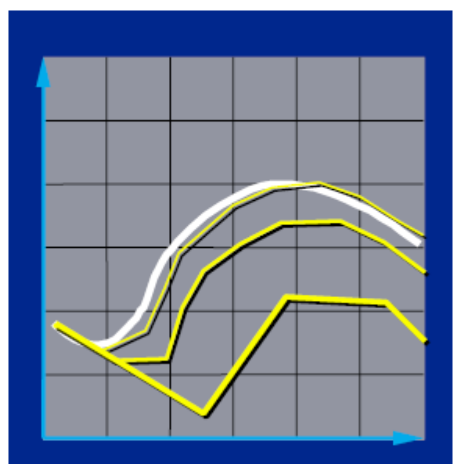


- Move a little step along the derivative to the next position
  - where h is the step size

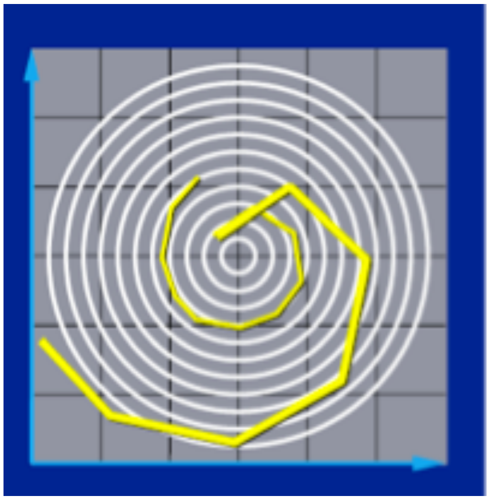
$$x(t_0 + h) = x(t_0) + h\dot{x}(t_0)$$

Euler Method and Step Size

- Simplest numerical solution
- Discrete time steps
- Bigger steps, bigger error

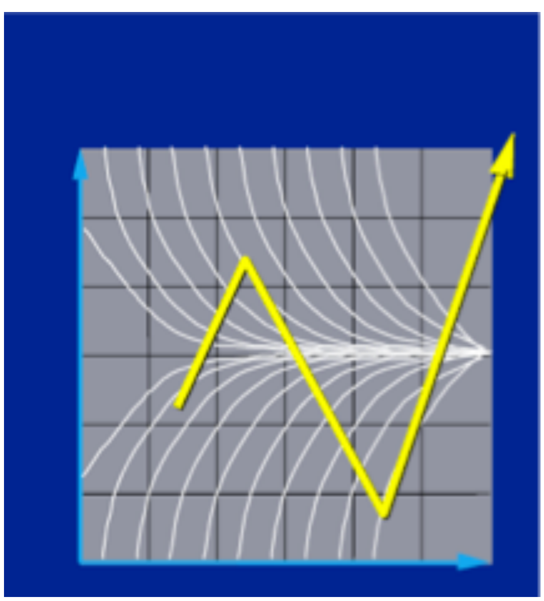


Accuracy



- Larger stepsize h leads to larger error
- Representation (round off) error will cause inaccuracy

Stability



- Euler method is unstable: solution might diverge!!

Beyond Euler

- Euler is a first order method
- We can improve the accuracy of each step if we extend to second derivatives
  - Based on Taylor series expansion

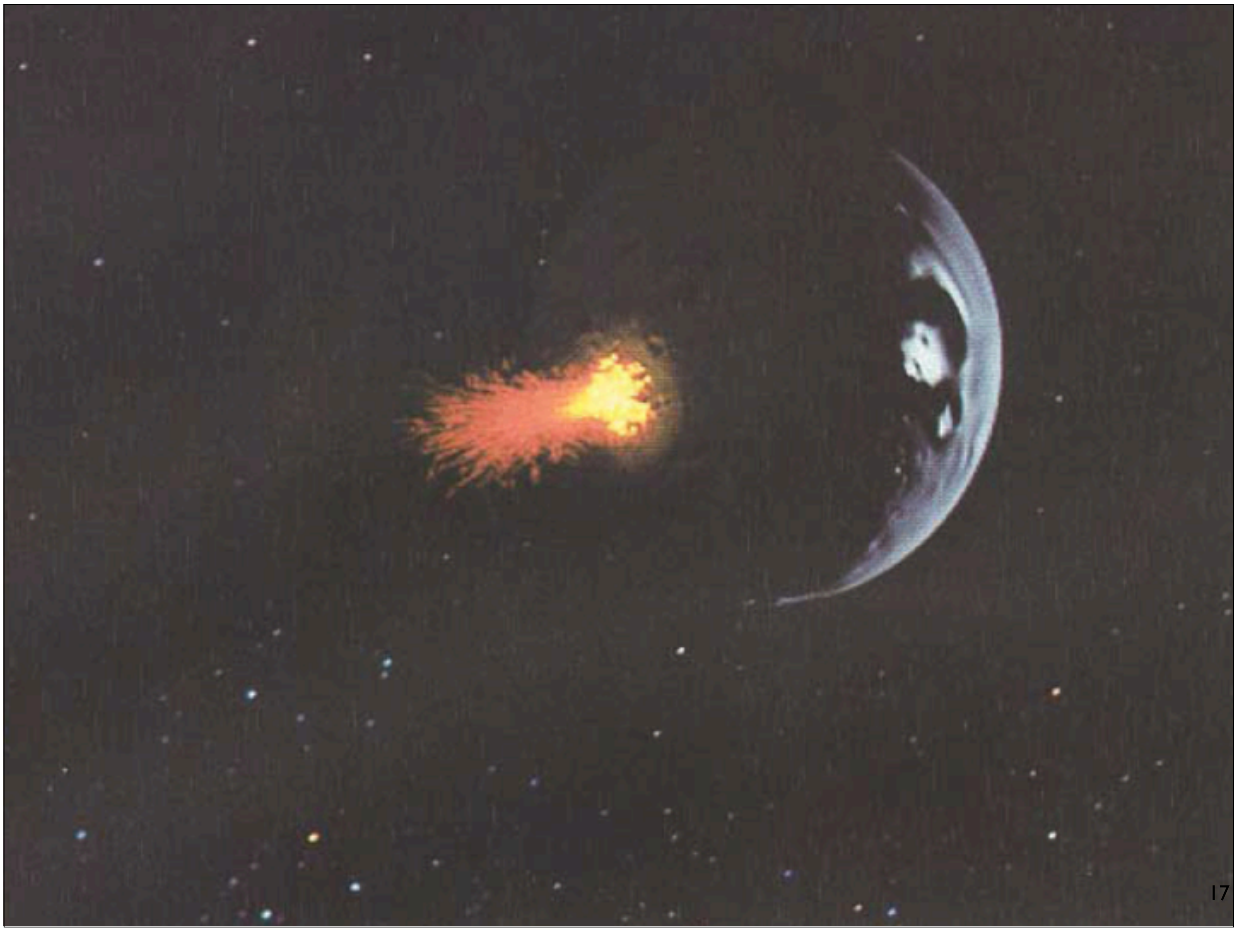
$$x(t_0 + h) = x(t_0) + h\dot{x}(t_0) + \frac{h^2}{2!}\ddot{x}(t_0) + \frac{h^3}{3!}\dddot{x}(t_0) + \ldots$$

- Euler: only first 2 terms
  - Error dominated by h^2

Bottom Line

- Use simpler methods if they get the job done

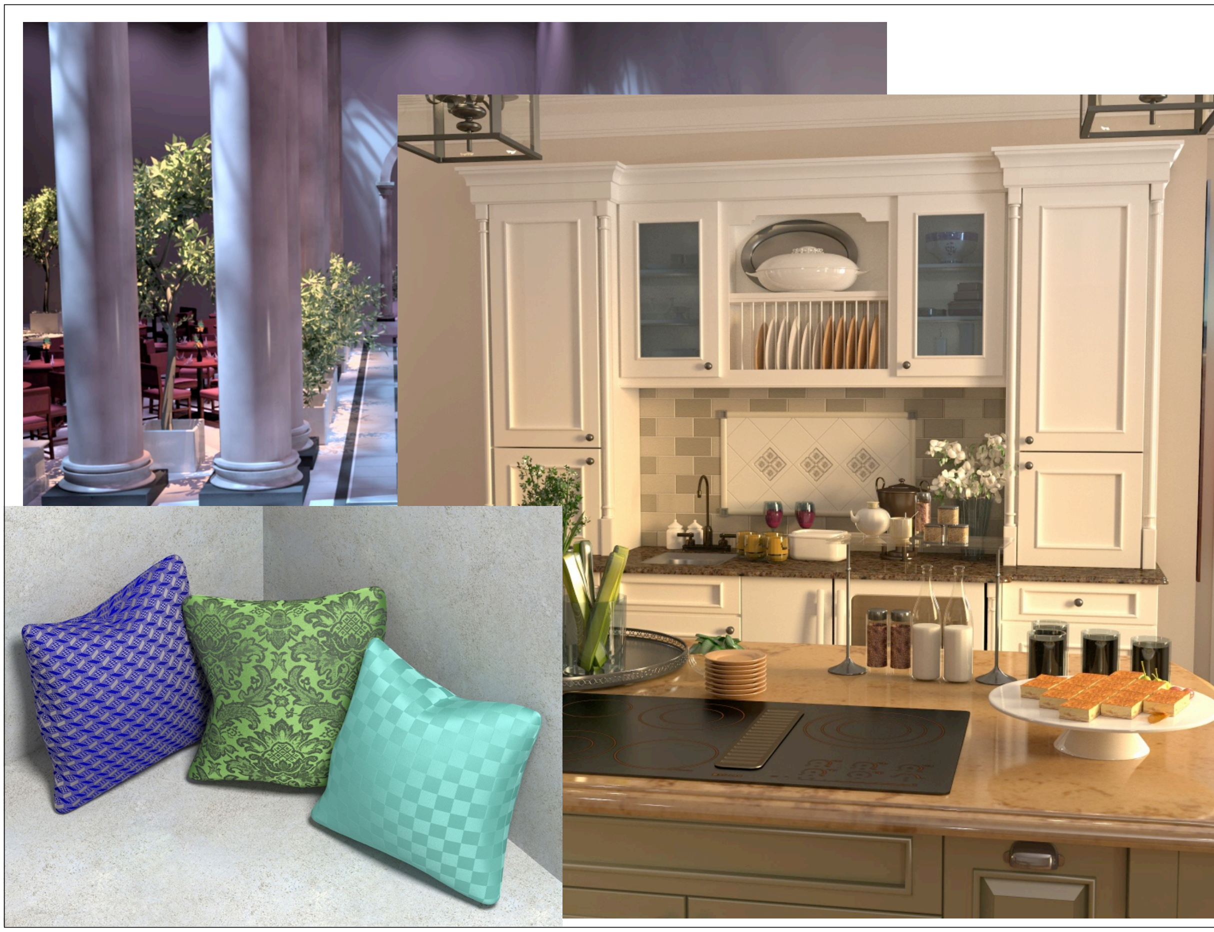
- In 4621 will discuss using particle systems that include simple physics



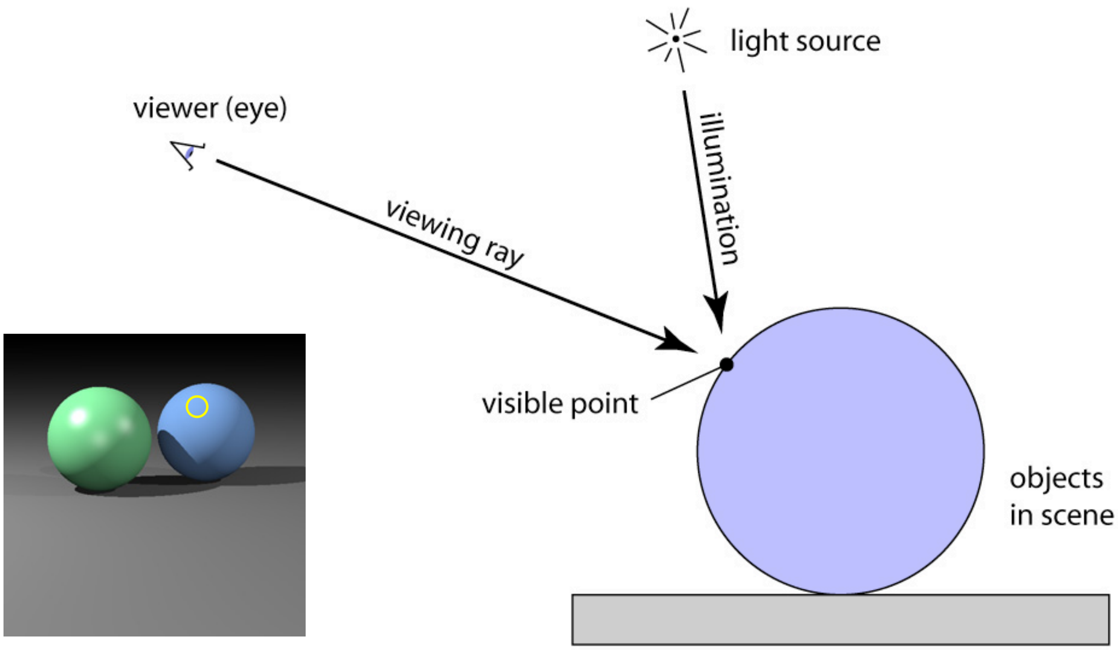
17

18

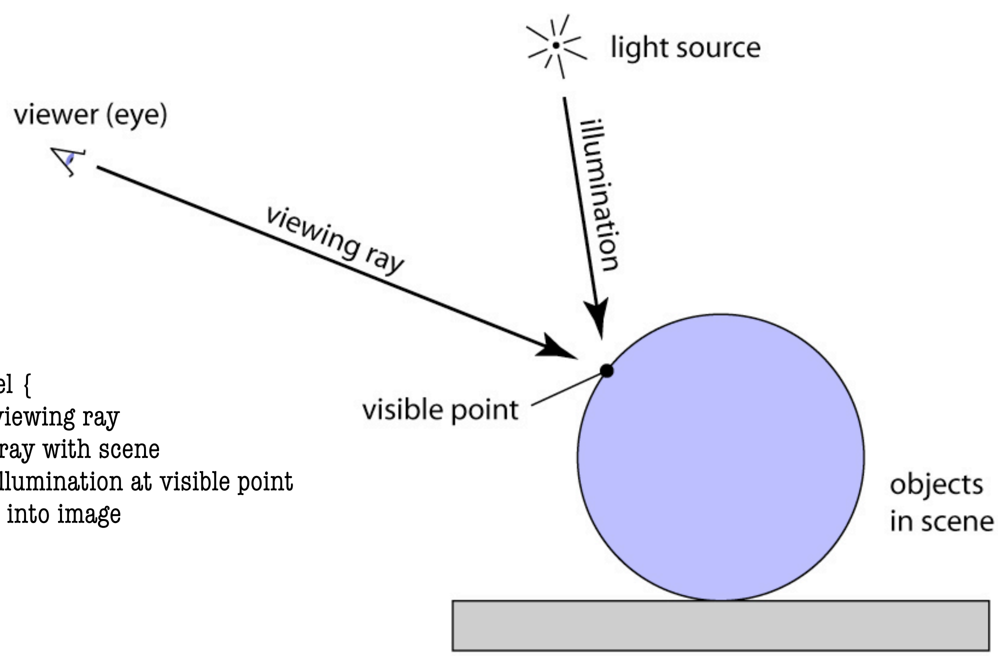
Ray Tracing



Ray tracing idea

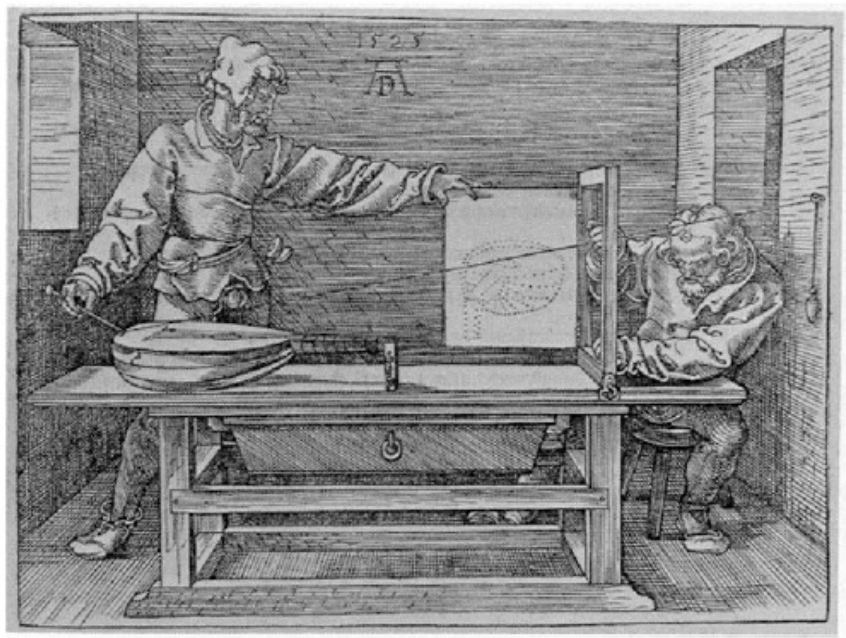


Ray tracing algorithm




```
for each pixel {
    compute viewing ray
    intersect ray with scene
    compute illumination at visible point
    put result into image
    }
```


Plane projection in drawing



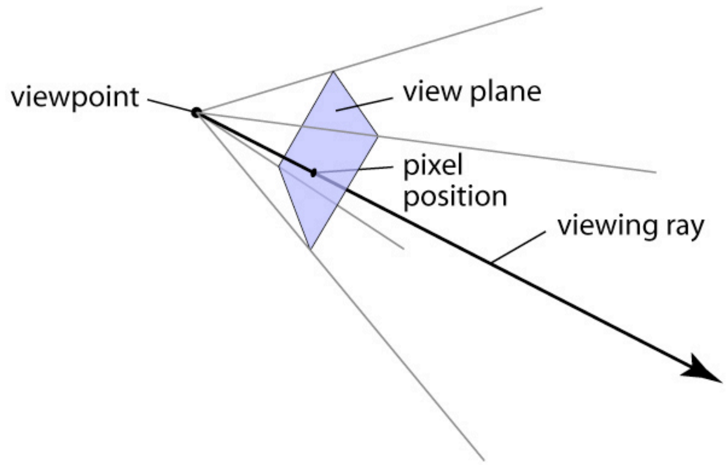
[Carlbom & Paciorek 78]

Ray generation vs. Projection

- Viewing in ray tracing
  - start with image point
  - compute 3D point that projects to that point using ray
  - do this using geometry
- Viewing by projection
  - start with 3D point
  - compute image point that it projects to
  - do this using transforms
- Inverse processes

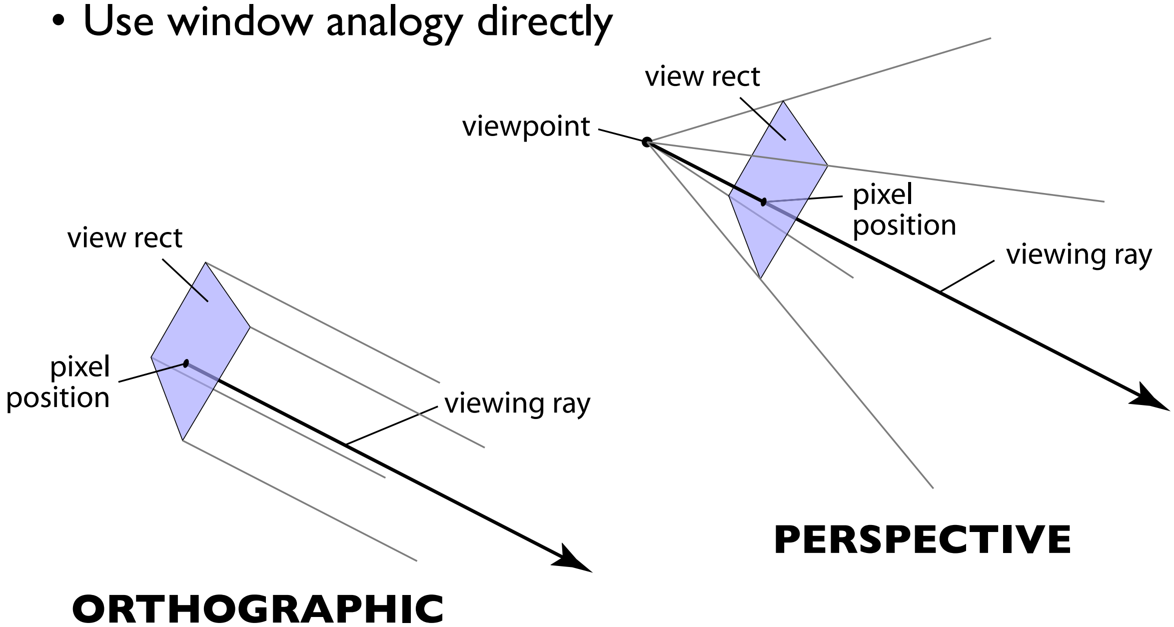
Generating eye rays

- Use window analogy directly



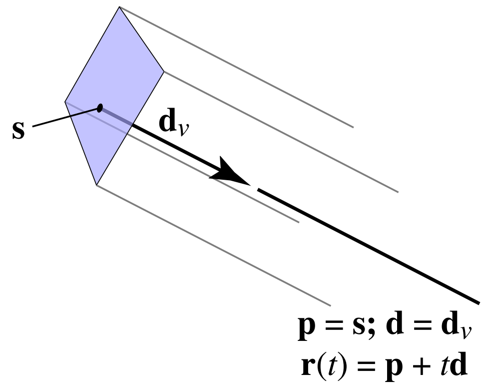
Generating eye rays

- Use window analogy directly



Generating eye rays—orthographic

- Just need to compute the view plane point **s**:

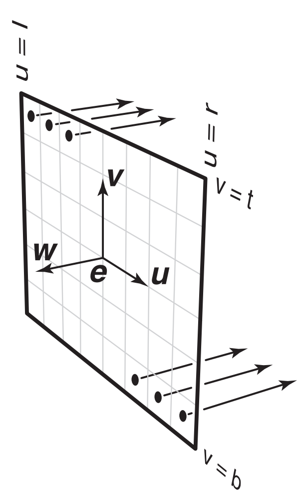


  – but where exactly is the view rectangle?

Generating eye rays—orthographic

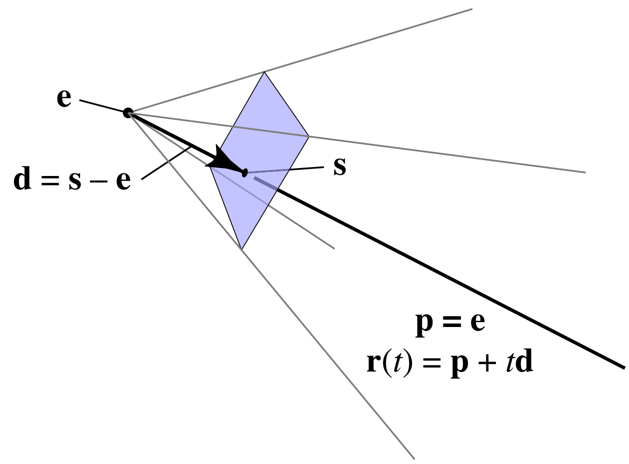
- Positioning the view rectangle
  – establish three vectors to be *camera basis:* **u**, **v**, **w**
  – view rectangle is in **u**–**v** plane, specified by l, r, t, b
  – now ray generation is easy:

$$\mathbf{s} = \mathbf{e} + u\mathbf{u} + v\mathbf{v}$$
$$\mathbf{p} = \mathbf{s};\ \mathbf{d} = -\mathbf{w}$$
$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$



Generating eye rays—perspective

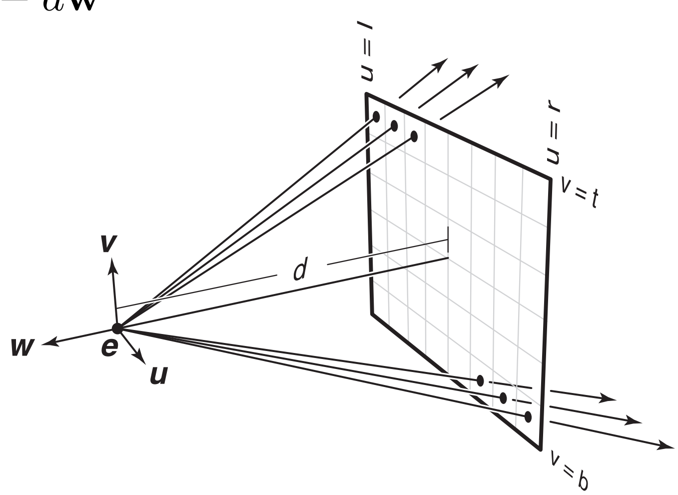
- View rectangle needs to be away from viewpoint
- Distance is important: "focal length" of camera
  - still use camera frame but position view rect away from viewpoint
  - ray origin always **e**
  - ray direction now controlled by **s**



Generating eye rays—perspective

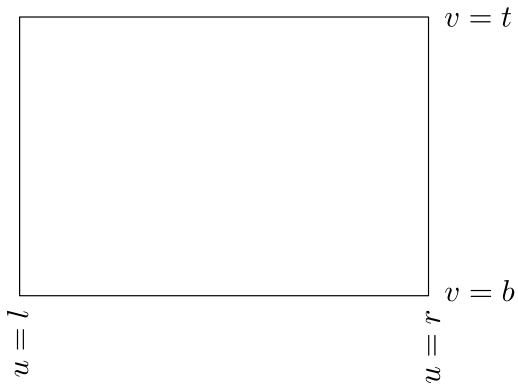
- Compute **s** in the same way; just subtract *d***w**
  - coordinates of **s** are (*u*, *v*, –*d*)

$$\mathbf{s} = \mathbf{e} + u\mathbf{u} + v\mathbf{v} - d\mathbf{w}$$
$$\mathbf{p} = \mathbf{e};\ \mathbf{d} = \mathbf{s} - \mathbf{e}$$
$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$



Pixel-to-image mapping

- One last detail: (*u*, *v*) coords of a pixel



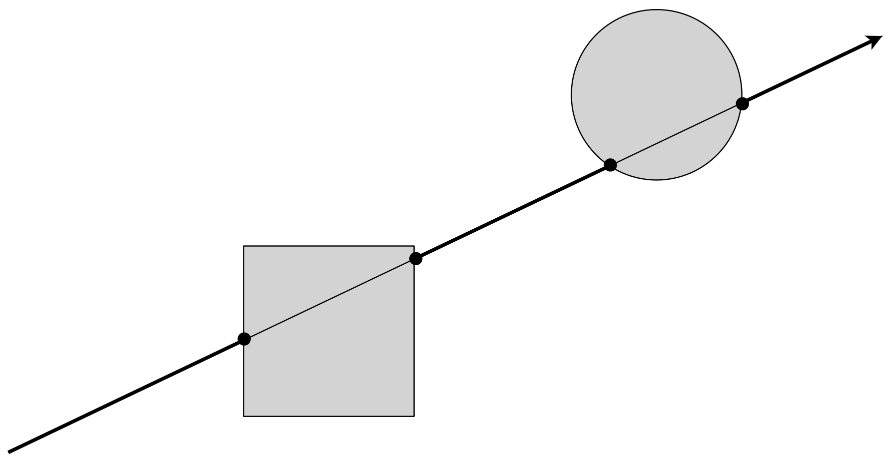
$$u = l + (r - l)(i + 0.5)/n_x$$
$$v = b + (t - b)(j + 0.5)/n_y$$

PA3A camera

- viewPoint == e
- projNormal == w, viewUp == up
  - Compute u, v from the above

- l = -viewWidth/2
- r = +viewWidth/2
- n_x = imageWidth

Ray intersection

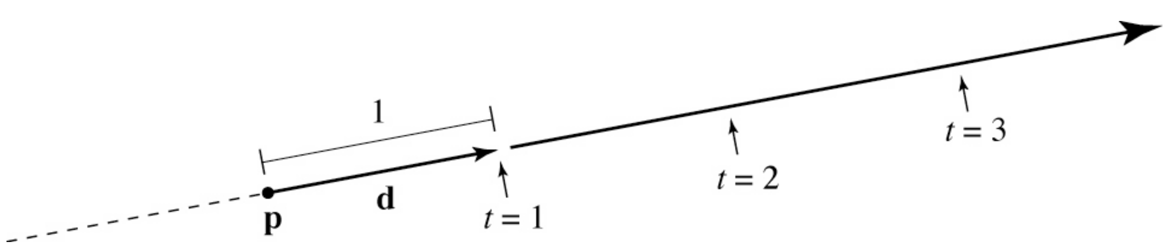


Ray: a half line

- Standard representation: point **p** and direction **d**

  $$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

  – this is a *parametric equation* for the line
  – lets us directly generate the points on the line
  – if we restrict to *t* > 0 then we have a ray
  – note replacing **d** with *a***d** doesn't change ray (*a* > 0)



Ray-sphere intersection: algebraic

- Condition 1: point is on ray

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

- Condition 2: point is on sphere
  - assume unit sphere; see Shirley for general

$$\|\mathbf{x}\| = 1 \Leftrightarrow \|\mathbf{x}\|^2 = 1$$

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{x} - 1 = 0$$

- Substitute:

$$(\mathbf{p} + t\mathbf{d}) \cdot (\mathbf{p} + t\mathbf{d}) - 1 = 0$$

  - this is a quadratic equation in *t*

Ray-sphere intersection: algebraic

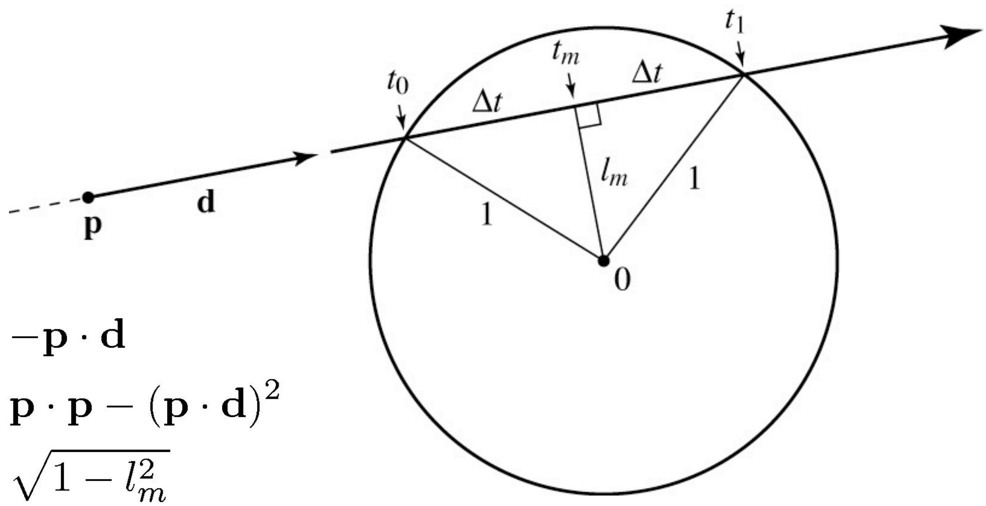
- Solution for *t* by quadratic formula:

$$t = \frac{-\mathbf{d} \cdot \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \mathbf{p})^2 - (\mathbf{d} \cdot \mathbf{d})(\mathbf{p} \cdot \mathbf{p} - 1)}}{\mathbf{d} \cdot \mathbf{d}}$$

$$t = -\mathbf{d} \cdot \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \mathbf{p})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

  - simpler form holds when **d** is a unit vector
    but don't necessarily assume this (for potential performance reasons)
  - discriminant intuition?
  - use the unit-vector form to make the geometric interpretation

Ray-sphere intersection: geometric



$$t_m = -\mathbf{p} \cdot \mathbf{d}$$

$$l_m^2 = \mathbf{p} \cdot \mathbf{p} - (\mathbf{p} \cdot \mathbf{d})^2$$

$$\Delta t = \sqrt{1 - l_m^2}$$

$$= \sqrt{(\mathbf{p} \cdot \mathbf{d})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

$$t_{0,1} = t_m \pm \Delta t = -\mathbf{p} \cdot \mathbf{d} \pm \sqrt{(\mathbf{p} \cdot \mathbf{d})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

Normal for sphere

Image so far

- With eye ray generation and sphere intersection

```
Surface s = new Sphere((0.0, 0.0, 0.0), 1.0);
for 0 <= iy < ny
   for 0 <= ix < nx {
      ray = camera.getRay(ix, iy);
      hitSurface, t = s.intersect(ray, 0, +inf)
      if hitSurface is not null
         image.set(ix, iy, white);
   }
```