Rasterization = sample pts to see if it lies inside triangle

Aliasing = high freq signals are undersampled (jaggies, wagon-wheel)

Antialiasing = removing high freq signals BEFORE sampling or increase sample rate

Linetest: use 3 line test to test if point inside △: $L(x,y) = -(x-x_0)(y_1-y_0)+(y-y_0)(x_1-x_0)$

$L(x,y) > 0$ = inside edge
$= 0$ = on edge
< 0 = outside edge

Point p {i+0.5f, j+0.5f}
framebuffer[y*width+i]

Box Blur ① select kernel size KxK ★SIMPLE & FAST
② for each pixel: replace value w/ avg of pixels in region
③ slide kernel across entire img
use nested for loop for implementation

Supersampling ① sample NxN locations within pixel & store to sample buffer
② Average out subpixels as that pixels value
artificially increase sampling rate above sampling freq

samplebuffer[(y*width+x)*sample_rate + (sample_y*sqrt_sample) + sample_x]
(second nested for loop)

argument
circle &c — check for null — vertex *√ — use dot notation → c.center — reference
use arrow → v->next — pointer

WINDING ORDER: vertices = CCW → check cross product
If cross product < 0 → clockwise → flip 2 vertices

Nyquist Thm: no aliasing from freq in signal < Nyquist freq
Nyquist freq: $f_{NYQUIST} = \frac{1}{2} f_{sampling}$ — no aliasing from freqs in signal less than Nyquist
can sample at 100Hz → Nyquist = 50Hz
$f_{sampling} > 2 f_{signal}$ — lowest freq you can sample w/ before aliasing
signal = 20Hz → sample above 40Hz

Transformations = functions on pts — RIGHT → LEFT

$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ TRANSLATE

$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ SCALE

$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ FLIP Y-AXIS

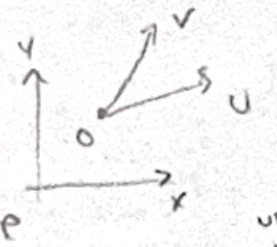
$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ ROTATE (x axis, y axis, origin)

vector → results.push_back(p) ← item

$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ SHEAR

$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ FLIP X-AXIS

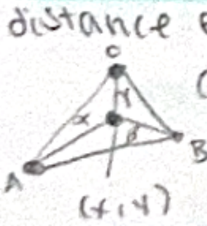
$\begin{bmatrix} u & v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} xu+yv+0 \\ 1 \end{bmatrix}$ coordinate system change



ISOMETRIC
rotations
translations
reflections

★ uniform scaling & rotation = commutative
★ rotate around origin

Barycentric coordinates = weighted distance from given point to vertices



$(x,y) = \alpha A + \beta B + \gamma C$
$\alpha + \beta + \gamma = 1$

★used for interpolation or **texture mapping**

each surface pt assigned texture (u,v) coordinate
$u = \alpha u_0 + \beta u_1 + \gamma u_2$
$v = \alpha v_0 + \beta v_1 + \gamma v_2$

close point = (higher coefficient)

$\alpha = \frac{-(x-x_B)(y_C-y_B)+(y-y_B)(x_C-x_B)}{-(x_A-x_B)(y_C-y_B)+(y_A-y_B)(x_C-x_B)}$

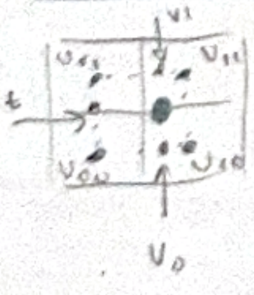
$\beta = \frac{-(x-x_C)(y_A-y_C)+(y-y_C)(x_A-x_C)}{-(x_B-x_C)(y_A-y_C)+(y_B-y_C)(x_A-x_C)}$

$\gamma = 1-\alpha-\beta$

★part of vertex + fragment processing

Nearest sampling = taking color of texel that is closest to barycentric coordinate

Bilinear sampling = weighted avg of 4 nearest texels



LERP(x, v0, v1) = v0 + x(v1 − v0)
2 horizontal lerp
u_0 = lerp(s, u00, u10)
u_1 = lerp(s, u01, u11)
Final vertical lerp: f(x,y) = lerp(t, u0, u1)

MIPMAPPING (¼ amt of memory)
Bilinear = ONE mipmap
Trilinear = LERP between two mipmap
0: 128×128 | 1: 64×64 | 2: 32×32 | 3: 16×16 | 4: 8×8 | ...
Big jump in texture space = far away → (low res) high level
small jump = close up → low level (high res)

① precompute lower res versions of texture
② store textures in mipmap
③ Adaptively choose mipmap level $D = \log_2 L$

$L = \max\left(\sqrt{\left(\frac{du}{dx}\right)^2+\left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2+\left(\frac{dv}{dy}\right)^2}\right)$

Rasterization Pipeline vertex processing → triangle → rasterization → fragment → framebuffer

[framebuffer operations]
Z-buffer = track depth
① initialize to ∞
store min z-value

Blinn-Phong Reflection Model
AMBIENT + DIFFUSE + SPECULAR = Phong reflection
★part of vertex + fragment processing

Ambient = constant color & shadows
↳ $k_a I_a$

Diffuse = independent of view direction
l = point light − source

$L_d = k_d\left(\frac{I}{r^2}\right)\max(0, n\cdot l)$ ← locate

Specular = intensity depends on view dir
v = e − s
$n = \text{bisector}(v,l) = \frac{v+l}{\|v+l\|}$
$L_s = k_s\left(\frac{I}{r^2}\right)\max(0, n\cdot h)^p$ ← shiny

Flat shading = shade each triangle
Gouraud shading = shade each vertex
Phong shading = shade each pixel

★Area weighted normal vectors at given vertex
① Iterate through incident faces ② Take $\frac{1}{2}\|a\|\|b\|\sin\theta$ (norm cross product)
③ Add up all areas & take unit vector

Cubic Hermite Spline = each cubic piece defined by: 2 points h_0, h_1; 2 tangents h_2, h_3

$P(t) = h_0 H_0(t) + h_1 H_1(t) + h_2 H_2(t) + h_3 H_3(t)$

$P(t) = [H_0(t)\ H_1(t)\ H_2(t)\ H_3(t)] \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$

$H_0(t) = 2t^3 - 3t^2 + 1$
$H_1(t) = -2t^3 + 3t^2$
$H_2(t) = t^3 - 2t^2 + t$
$H_3(t) = t^3 - t^2$

Catmull Rom Spline → passes through all give points except first and last
Given $h_0\ h_1\ h_2\ h_3$: interpolate cubic hermite spline on h_1 and h_2
estimate tangents: $\frac{1}{2}(h_2 - h_1) = v_1$, $\frac{1}{2}(h_3 - h_1) = v_2$

$P(t) = [H_0(t)\ H_1(t)\ H_2(t)\ H_3(t)] \begin{bmatrix} h_1 \\ h_2 \\ v_1 \\ v_2 \end{bmatrix}$

Bezier curve Given $P_0 \cdots P_n$
compute $B(t) = \sum_{i=0}^{n} B_i^n(t) P_i$ ↕ Bezier basis

De Casteljau's Algorithm
compute intermidate control points through LERP: $P_i' = (1-t)P_i + t(P_{i+1})$

$b_0'(t) = (1-t)b_0 + tb_1$
$b_1'(t) = (1-t)b_1 + tb_2$
$b_0^2(t) = (1-t)b_0' + tb_1'$

Bezier surfaces = extend curves to surfaces (u,v) eval de Casteljau w/ point u, eval v on curve

camera(e, U, V) — up
eye point
view

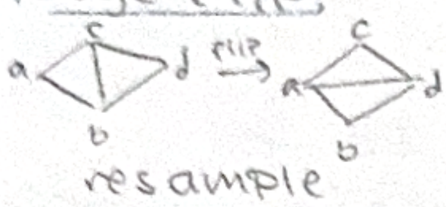
$\begin{bmatrix} r_x & u_x & -v_x & e_x \\ r_y & u_y & -v_y & e_y \\ r_z & u_z & -v_z & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$ transform matrix

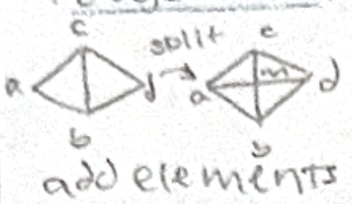
C0 continuity: ends of segments meet

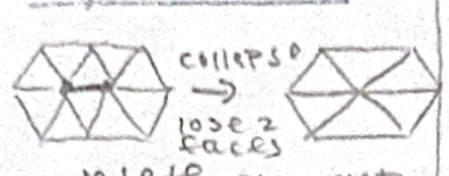
C1 continuity: straight line control pts → Equal tangent for both segments

C2 continuity: same 2nd derivative @ joint

Half edges
data structure used to represent mesh
- 2 half-edges = "glue" between mesh elements
- →twin and →next to traverse & process

Edge Flip


Edge Split


Edge collapse


Loop subdivision
- n = vertex degree

① Split each triangle into 4

② Update old & new vertex positions as weighted sum

$$v'_{old} = (1-nu)v_{old} + \sum_{v_j \in N(v_{old})} uv_j \qquad v'_{new} = \frac{3}{8}(v_{left}+v_{right}) + \frac{1}{8}(v_{up}+v_{down})$$

$$u = \frac{3}{16} \text{ if } n=3, \quad \frac{3}{8n} \text{ otherwise}$$

```
HalfedgeIter h = v->halfedge()
do {
  HalfedgeIter h_twin = h->twin
  VertexIter v = h_twin->vertex
  h = h_twin->next
} while(h != v->halfedge())
```

③ for any new edge that connects new → old vertex
- Edge Flip if each non quad face ~/ n edges → extraordinary pt of degree n and n quad faces

Catmull Clark subdivision
Downsample by collapsing edges → choose lowest quadric error
- ① add vertex in face
- ② add midpoint on each edge
- ③ connect
- *meshes w/ variable polygon

↳ sum of distances to planes

Ray Tracing
= light rays for photorealistic rendering

Recursive Ray Tracing: bounce from 1 object to another until max level or non-specular

Ray equation $r(t) = o + td$ (o: origin, d: unit direction)

Plane equation: $(p-p')\cdot N = 0$ (p': any pt on plane, N: normal vector)

Ray-plane-intersection: $p = r(t) \rightarrow (o+td-p')\cdot N = 0$

$$t = \frac{(p'-o)\cdot N}{d\cdot N}$$

- $t<0$: invalid intersection
- $d\cdot N = 0$ and $(o-p')\cdot N = 0$ → infinite
- $(o-p')\cdot N \neq 0$ → zero

Ray-surface intersection
Given $f(x,y,z)$ and $r(t)$

$$f(x,y,z) = \frac{(x-2)^2}{4} + (y-2)^2 + \frac{z^2}{4} - 1 \qquad r(t) = (0,0,0) + t(1,1,0)$$

$$\left.\begin{aligned} x &= 0+t\cdot 1 = t \\ y &= 0+t\cdot 1 = t \\ z &= 0+t\cdot 0 = 0 \end{aligned}\right\}$$

*take smallest t

$$f(x,y,z) = \frac{(t-2)^2}{4} + (t-2)^2 - 1 = 0 \rightarrow t = 2 \pm \sqrt{\frac{4}{5}}$$

plug back into ray eq: $r(t) = (0,0,0) + (2-\sqrt{\frac{4}{5}})(1,1,0)$

$= \left(2-\sqrt{\frac{4}{5}}, 2-\sqrt{\frac{4}{5}}, 0\right)$ first intersection point

Ray-object intersection
① Set $p = o + td$

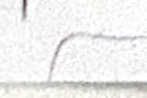
② Find t where $f(p) = f(o+td) = 0$

sphere: $p: (p-c)^2 - R^2 = 0$

$(o+td-c)^2 - R^2 = 0$

$$t = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

axis aligned ray plane intersection

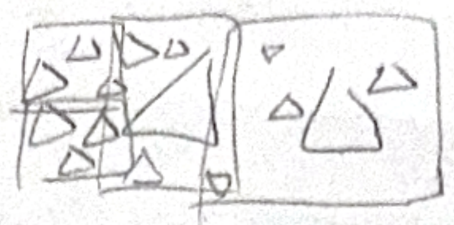


$$t = \frac{p'_x - o_x}{d_x}$$

Acceleration structures


Internal Nodes:
- ① Bounding box
- ② Reference to children

Leaf Nodes
- ① Bounding box
- ② List of primitives in the box



Intersect(Ray ray, BVH node)
- if (ray misses node.bbox) return
- if (node is leaf node) test intersection w/ all objects; return closest intersection
- hit1 = Intersect(ray, node.left)
- hit2 = Intersect(ray, node.right)
- return closer(hit1, hit2)

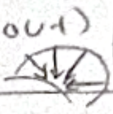
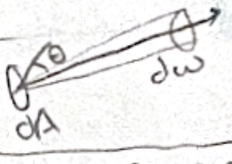
① Always pick longest axis to divide

② pts should be sorted by position

③ Use center of mass

④ BVH should be balanced

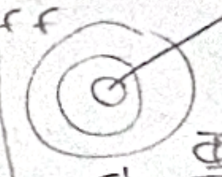
Radiometry & Photometry
$I = \frac{I_0 \cdot \cos\theta}{d^2}$ (d: distance; solid angle → 2π; 1000 lum)

Name	Radiometry	Photometry
Energy: $Q = \int_{t_0}^{t_1} \Phi \, dt$	Radiant energy Joules (W·s)	Luminous Energy Lumen·sec
Flux: $\Phi = \frac{dQ}{dt}$	Radiant power W	Luminous power Lumen (Candela·sr)
I: Angular flux density $= \frac{d\Phi}{dw}$	Radiant Intensity W/sr 	Luminous Intensity Candela (Lumen/sr)
Spatial flux density $E(p) = \frac{d\Phi(p)}{dA}$	Irradiance (in) radiosity (out) W/m² 	Illuminance (in) luminosity (out) Lux (Lumen/m²)
Spatio-angular flux density $L(p,w) = \frac{d^2\Phi(p,w)}{dw\,dA\cos\theta}$ $= \frac{dE(p)}{dw\cos\theta} = \frac{dI(p,w)}{dA\cos\theta}$	Radiance W/m²/sr 	Luminance Nit Candela/m²

Lambert cosine law
Irradiance @ surface ~~proportional~~ to cosine of angle between light & surface normal

$$E = \frac{\Phi}{A}\cos\theta$$

Irradiance falloff



Monte Carlo integration
= estimate integral w/ rand sampling

✓ high dim, gen function

✗ noise, slow to converge

$$F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}$$

*can lead to lower variance/noise

$$E[F_N] = \int_a^b f(x)\,dx$$

*UNBIASED!

$x \sim p(x)$, $p(x) \geq 0$, $\int p(x)\,dx = 1$, $E[X] = \int x p(x)\,dx$

Inversion method
Draw rand samples from PDF

① Calculate PDF $p(x) = cf(x)$

② Calculate CDF $P(x) = \int_a^x p(x)\,dx$

③ Solve $x = P^{-1}(\xi)$

① $P(x) = \alpha x$, $1 = \int_0^2 p(x)\,dx = \alpha\int_0^2 x\,dx = 2\alpha$, $p(x) = \frac{1}{2}x$

② $F(x) = \int_0^x \frac{1}{2}t\,dt = \frac{t^2}{4}\Big|_0^x = \frac{x^2}{4}$

③ $U = F(x)$ → solve for x: $t = \frac{x^2}{4}$, $4t = x^2$, $2\sqrt{t} = x$

Solid angle
$\Omega = \frac{A}{r^2}$

Sphere = 4π steradians

hemisphere = 2π

$dA = r\,d\theta\,r\sin\theta\,d\phi$, $dw = \frac{dA}{r^2}$, $= \sin\theta\,d\theta\,d\phi$

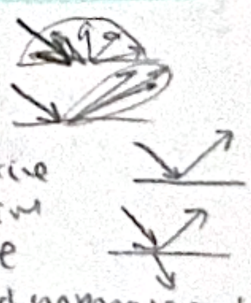
Light transport
w_i = dir of incoming

w_o = dir of outgoing

BRDF = amt of light from w_i reflected in w_o

$$L_r(p,w_o) = L_e(p,w_o) + \int_{H^2} f_r(p,w_i,w_o)L_i(p,w_i)\cos\theta_i\,dw_i$$

Global illumination
- Diffuse
- Glossy
- Ideal reflective
- Ideal reflection + refractive



concentrated normals = glossy

spread out normals = diffuse

$f(i,o) = F(i,h)\,G(i,o,h)\,D(h) / 4(n,i)(n,o)$

- F(i,h): changes w/ viewing angle (Shell Scivolo)
- G(i,o,h): shadowing masking
- D(h): dist of normals anisotropy

light

Russian Roulette: randomly terminate current ray w/ $P = 1-P_{RR}$, $X_{rr} = \frac{L'}{P_{RR}}$ (0 if terminated)

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} r_S \cdot S_R & r_S \cdot S_G & r_S \cdot S_B \\ r_M \cdot S_R & r_M \cdot S_G & r_M \cdot S_B \\ r_L \cdot S_R & r_L \cdot S_G & r_L \cdot S_B \end{bmatrix}^{-1} \begin{bmatrix} -r_S- \\ -r_M- \\ -r_L- \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

Chromaticity Diagram = pure saturated spectral at corners
- desaturated in center
- NO BLACK

CIELAB = strives for perceptual uniformity

Virtual Reality (both vergence + accomodation provide depth cues)

vergence = rotation of eyes to focus on near/far objects

accomodation = eye's ability to change shape of lens to bring objects into focus
- in VR: lens accomodate physical screen distance NOT virtual depth

vergence-accomodation conflict = vergence cues of virtual distance, accomodation cues of display distance

Mismatch signals = visual discomfort

VR needs low latency tracking/rendering

monoscopic = both eyes w/ same 360 frame
↳ NO stereoscopic depth

SENSORS

- photons enter according to Poisson distribution
  ↳ occurence of indep events over fixed time interval
  ↳ **avg rate of arrival = λ**

$$P(X=x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Low light = fewer photons arrive

Russian Roulette = finite time algo that gives UNBIASED MC estimate to path tracing integral which normally has infinite recursion
- randomly terminate w/ probability P
- do Not terminate → weight radiance returned by recursive call by $\frac{1}{1-P}$

Material Modeling

Anisotropic materials: directionality of underlying surface
↳ NOT same in every dir (ie elongated curve specular highlight)

Specular refraction = light transmitted through surface when it's in new medium

microfacet material: concentrated normals → GLOSSY (smaller specular)
spread out normals → DIFFUSE (larger specular)

$f=ma, f=-kx$

Foveated rendering = rendering at full detail where user looking & lower detail elsewhere in peripheral (reduces perceived latency)

inside out tracking = tracking sensors on headset → look outward

virtual reality = fully immersed in virtual world

augmented reality = real world + digital objects w/ optical/video pass

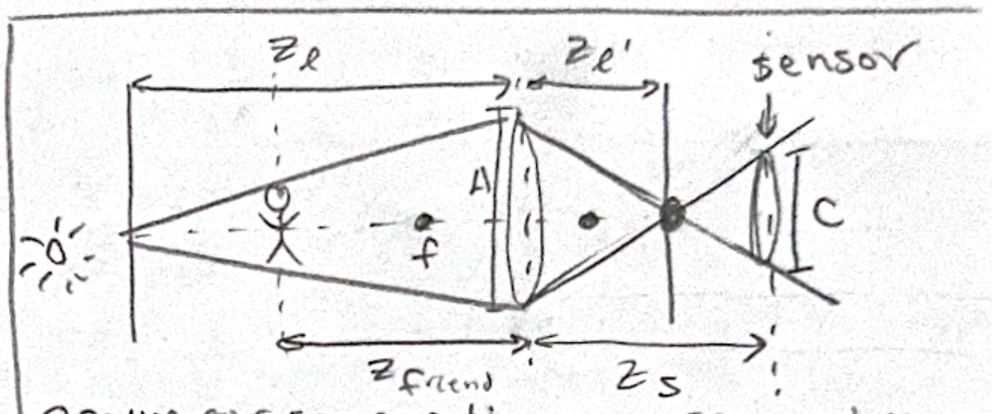
BRDF = how much light reflected into each outgoing direction w_r from each incoming direction

$$L_r(p, w_r) = \int_{H^2} f_r(p, w_i \to w_r) L_i(p, w_i) \cdot \cos\theta_i \, dw_i$$

Estimator = $\frac{1}{N} \sum_{j=1}^{N} \frac{f_r(p, w_j \to w_i) L_i(p, w_j) \cos\theta_j}{P(w_j)}$

```
DirectLighting(p, wo):
    wi, pdf = p.brdf.sampledir(wo)
    if(scene.shadow(p, wi))
        return 0
    else:
        Li = lights.radiance(intersect(p, wi), -wi)
        return Li * p.brdf(wi, wo) * costheta / pdf;
```

Judder = mismatch between head motion & image → read head pos as late as possible



① solve for sensor distance (focused on friend)

$$\frac{1}{z_\ell} = \frac{1}{f} - \frac{1}{z_s} \to z_s = \frac{1}{\frac{1}{f} - \frac{1}{z_\ell}}$$

② solve for light distance (where light in focus)

$$\frac{1}{z_{\ell'}} = \frac{1}{f} - \frac{1}{z_\ell}$$

(if $z_{\ell'} < z_s$, circle of confusion is after where rays converge)

③ solve for circle of confusion

$$C = A \cdot \frac{z_s - z_{\ell'}}{z_{\ell'}} = \frac{f}{N} \cdot \frac{z_s - z_{\ell'}}{z_{\ell'}}$$

$$\frac{c}{A} = \frac{d'}{z_{\ell'}} = \frac{|z_s - z_{\ell'}|}{z_{\ell'}}$$

f# ↗

$$A = \pi\left(\frac{D}{2}\right)^2 = \frac{\pi D^2}{4} = \frac{\pi}{4}\left(\frac{f}{f\text{-num}}\right)^2 = \frac{\pi f^2}{4(f\text{-num})^2}$$

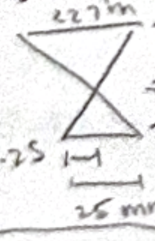
- turning on 1 pixel and measuring each wavelength = emission spectra of that single pixel
- color matching = take in spectral emission values and outputting RGB (convert spectral to RGB (convert RGB to spectral (RGB)))

$S_{disp} = R_1 S_{r_1} + G_1 S_{G_1} + B_1 S_{B_1}$

- focus on nearer = decreased depth of field
- increase focal length = zoom in
- MoCap uses multiple cameras, markers, triangulation, w/ INFRARED
- cubic Bezier curve (degree 3, 4 control pts)

$B_0(t) = (1-t)^3$
$B_1(t) = 3t(1-t)^2$
$B_2(t) = 3t^2(1-t)$
$B_3(t) = t^3$

FOV question:



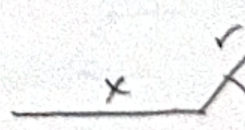
$\frac{22.7}{x} = \frac{25}{2.50}$

① Desired magnification = 0.5 $\frac{1}{2} = \frac{z_i}{z_o} \to z_i = \frac{z_o}{2}$ *when z_o NOT INF

② apply thin lens: $\frac{1}{f} = \frac{1}{z_i} + \frac{1}{z_o} = \frac{2}{z_o} + \frac{1}{z_o} = \frac{3}{z_o}$

4x the area = 4x as many photons

2x as many sensor pixels = 2x as many photons per pixel



$p = (a, b) = (x + r\cos\theta, r\sin\theta)$

PDF of sample from hemisphere = $\frac{1}{2\pi}$

$\int \sin x = -\cos x$

$\int \cos x = \sin x$

$z = \int_{-\pi/2}^{\pi/2} \cos(x)\,dx = \sin x \Big|_{-\frac{\pi}{2}}^{\frac{\pi}{2}} = 1 + 1 = 2 \to \frac{1}{2}\cos x \to \int_{-\pi/2}^{x} \frac{1}{2}\cos t\,dt = \frac{1}{2}(\sin x + 1)$

- sample $g(w_i)$ from S^2 ↳ normalize by $\int_{S^2} g(w_i)\,dw$

$$\frac{1}{N}\sum_{i=1}^{N} \frac{F(P_{-1}(\frac{X_i}{\pi} + \frac{1}{2}))}{P_G(X_i/\pi + \frac{1}{2}))\pi}$$

$u = \frac{1}{2}(\sin(x) + 1) \to x = \sin^{-1}(2u - 1)$

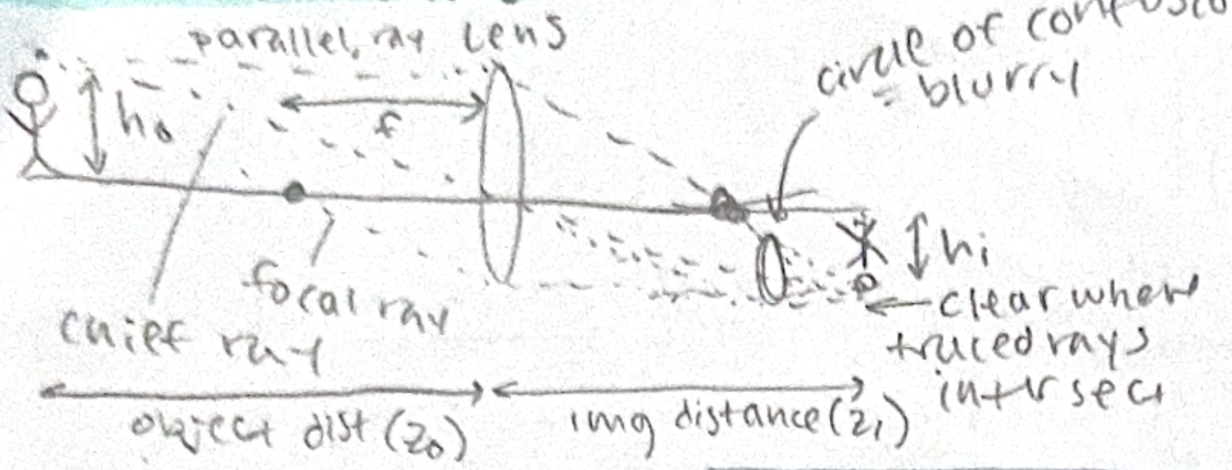
$(\text{dot}(n, p) + d)^2 = (n \cdot p + d)^2$

$= (n_x x + n_1 y + n_2 z + d)^2$

= outer products

illuminance E from isotropic: $E = \frac{X}{4\pi r^2}$

cameras & lenses

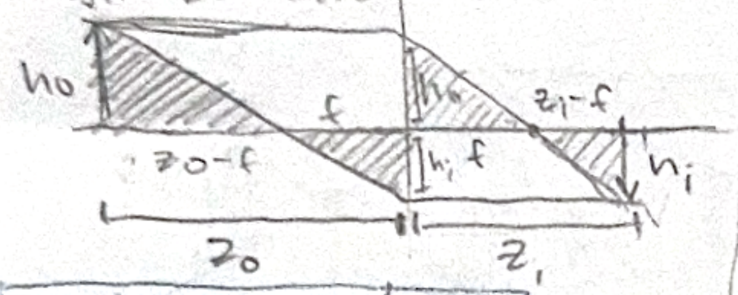


thin lens equation $\frac{1}{f} = \frac{1}{z_o} + \frac{1}{z_i}$

thin lens approximation = assume all parallel rays entering lens pass through its focal pt

magnification $M = \frac{h_i}{h_o} = \frac{z_i}{z_o}$

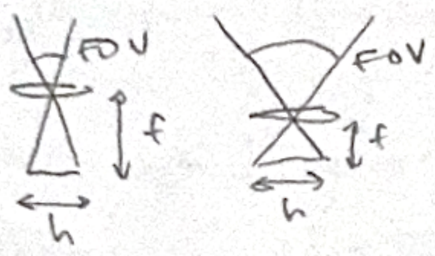
* increase focal length = zoom in



$\frac{h_o}{z_o - f} = \frac{h_i}{f}$ $\quad \frac{h_o}{f} = \frac{h_i}{z_i - f}$

$z_i < z_o$: object larger than img
$z_i = z_o$: 1:1 macro imaging
$z_i > z_o$: img larger than object

Field of view/FOV = angle of scene captured by camera lens



$FOV = 2\arctan\left(\frac{h}{2f}\right)$ → sensor height, → focal length

** can use similar triangles with $\frac{h}{f}$

small focal length → larger FOV
small sensor size → small FOV
small aperture → more depth of field
small aperture → big f stop → less exposure
big f-stop

Depth of field = range of object depths rendered w/ acceptable sharpness

FStop # = $\frac{\text{focal length}}{\text{diameter of aperture}}$

exposure = irradiance × time × gain

larger aperture = higher irradiance
↑ opening in lens
exposure $\propto \left(\frac{1}{f\#}\right)^2$

f stop #s increase by factors of $\sqrt{2}$
large aperture = small f stop #

$\sqrt{2}$: increase in aperture → 2× amt of light
×2 shutter duration → ×2 exposure
×2 ISO gain → ×2 exposure
−1 f stop → ×2 exposure

ISO = sensor's light sensitivity
high ISO → more noise

*optically sharper = preserves higher detail

SIMULATION

conditionally stable

Forward Euler's Method $\frac{dx}{dt} = f(x,t)$

x^t = position
$\dot{x}^t$ = velocity
$\ddot{x}^t$ = acceleration

$x^{t+\Delta t} = x^t + \Delta t\, \dot{x}^t$
$\dot{x}^{t+\Delta t} = \dot{x}^t + \Delta t\, \ddot{x}^t$

*NOT UNCONDITIONALLY

✓ simple, easy to compute
✗ errors accumulate (small stepsize = lower error)
✓ more stable not stable

Implicit/Backwards Euler

$x^{t+\Delta t} = x^t + \Delta t\, \dot{x}^{t+\Delta t}$
$\dot{x}^{t+\Delta t} = \dot{x}^t + \Delta t\, \ddot{x}^{t+\Delta t}$

✓ unconditionally (more) stable
✗ difficult nonlinear eq

Modified Eulers

$x^{t+\Delta t} = x^t + \frac{\Delta t}{2}(\dot{x}^t + \dot{x}^{t+\Delta t})$ ✓ more stable

$\dot{x}^{t+\Delta t} = \dot{x}^t + \Delta t\, \ddot{x}^t$

Verlet Integration

$x^{t+\Delta t} = x^t + \Delta t\, \dot{x}^t + \frac{1}{2}(\Delta t)^2 \ddot{x}^t$

$\dot{x}^{t+\Delta t} = \frac{x^{t+\Delta t} - x^t}{\Delta t}$

✗ Dissipates energy
✓ constrain positions to prevent unstable behavior

Hooke's Law $f = -kx$

$f_{a \to b} = k_s \frac{b-a}{\|b-a\|}(\|b-a\| - \ell)$

k_s = spring constant; ℓ = length

$f_{b \to a} = -f_{a \to b}$

+/2	f/2.8	F/4	F/5.6	F/8	F/11	F/16
most blurry background	←				→	least blur background

Animation

forward kinematics = input: angles for joints
↓
output: computer determines final positio

inverse kinematics = input: ending po
output: joint angle to reach that
x rarely closed form
↳ use gradient descent

keyframes = important moments in some transition/motion between start/end

skinning = move surface along w/ bones

1. squash/stretch
2. anticipation
3. staging
4. Followthrough
5. Exaggeration
6. Ease in/ease out
7. Arcs
8. secondary action
9. Timing
10. Personality
11. Appeal

LIGHT

= characterized by SPD

spectral power distribution = power in light beam @ given wavelength watts/nm

monospectrum = near single wavelength/color

Total = $R s_R(\lambda) + G s_G(\lambda) + B s_B(\lambda)$
↑ red brightness level ← red monospectrum

$\begin{bmatrix} 1 & 1 & 1 \\ s_R & s_G & s_B \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$

L = orange yellow
M = green yellow
S = blue

cone cells

cone cell responses

$L = \int r_L(\lambda)\, s(\lambda)\, d\lambda$
$M = \int r_M(\lambda)\, s(\lambda)\, d\lambda$
$S = \int r_S(\lambda)\, s(\lambda)\, d\lambda$

$r_S(\lambda)$ ↑ spectral response for cone cell; $s(\lambda)$ SPD

$\begin{bmatrix} - r_S - \\ - r_M - \\ - r_L - \end{bmatrix} \begin{bmatrix} 1 \\ S \\ 1 \end{bmatrix}$

metamer = 2 DIFFERENT SPD w/ same visual (L,M,S) cone response

display color → human vision ↑ cone spectral response → DISPLAY monospectral distributions → display values → real life color → SPD of real color

$\begin{bmatrix} - r_S - \\ - r_M - \\ - r_L - \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ S_R & S_G & S_L \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} - r_S - \\ - r_M - \\ - r_L - \end{bmatrix} \begin{bmatrix} 1 \\ S \\ 1 \end{bmatrix}$

Gamut = range of colors that can be displayed on device → device dependent

* color matching = linear
* 3 primary colors for normal color vision

rods = primarily in low light → diff shades of gray
cones = sensation of color

Hue, Saturation, value

Hue = dominant wavelength
saturation = how vivid color is
value = amt of light
* MORE INTUITIVE