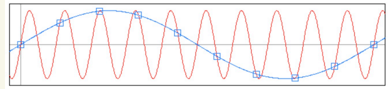
Rasterization = sample points to see if inside triangle

Aliasing = high freq signals are under-sampled
↳ can resemble continuous signal of lower freq



Anti-aliasing = removing high freq signals before sampling

Box blur =
1. select kernel size k×k
2. for each pixel in img, replace its value w/ AVERAGE of all pixel values in kernel region
3. slide kernel across entire img

★ simple but FAST

Supersampling = artificially increase sampling rate above sampling frequency

Winding order = order of vertices of △ → convention is COUNTER clockwise

★ If neg cross product → CW order → swap 2 vertices

Nyquist freq = $f_{nyquist} = \frac{1}{2} f_{sampling}$ → NO aliasing from freqs in signal that are LESS than Nyquist freq

★ If device can sample at 100Hz → Nyquist = 50Hz

$f_{sampling} > 2 f_{signal}$, $f_{sampling}$ = lowest freq you can sample w/ before aliasing

★ If signal = 20Hz → sampling ABOVE 40 Hz

Transformations

Homogeneous coordinates = represent point in n-dim space w/ n+1 coordinates

$(x, y)^T \to (x, y, w)^T$ w=1 → point
$(x, y, z)^T \to (x, y, z, w)^T$ w=0 → vector

} allows us to represent points & vectors in same coordinate system

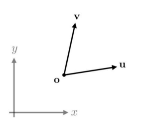
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
Translate

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

$$\begin{bmatrix} \mathrm{u} & \mathrm{v} & \mathrm{o} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\mathrm{u} + y\mathrm{v} + \mathrm{o} \\ 1 \end{bmatrix}$$
Coordinate System Change

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
Rotate

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



Flip across y-axis
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Flip across x-axis
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Isometric = preserve distances between every pair of points on object

✓ Rotations, translations, reflections
✗ Scaling, shearing

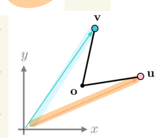
★★ Transformation matrices multiplied RIGHT to LEFT

$[T_3][T_2][T_1] = [\text{Final } T]$

Coordinate system transformation: (o, u, v) to (x, y)
(o = origin; u, v = 2 unit vectors)

Point (1,0)
$$\begin{bmatrix} \mathrm{u} & \mathrm{v} & \mathrm{o} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathrm{u} + \mathrm{o} \\ 1 \end{bmatrix}$$

Point (0,1)
$$\begin{bmatrix} \mathrm{u} & \mathrm{v} & \mathrm{o} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathrm{v} + \mathrm{o} \\ 1 \end{bmatrix}$$



Camera transformation:

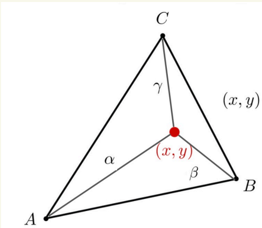
camera (e, u, v) — e = eye point, u = up, v = view

Transform matrix

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

Barycentric coordinates

= weighted distance from given point to vertices



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

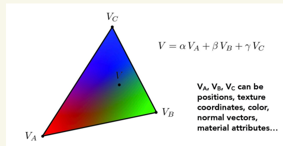
$\alpha + \beta + \gamma = 1$

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

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

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

* used for interpolation



Texture mapping

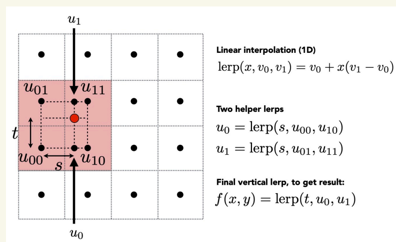
= moving from screen space to texture space (u, v)

* can interpolate texture coordinates (u, v) w/ $u = \alpha u_0 + \beta u_1 + \gamma u_2$
  $v = \alpha v_0 + \beta v_1 + \gamma v_2$

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

Bilinear sampling = weighted avg of 4 nearest texels w/ 3 LERPs

↳ $\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$

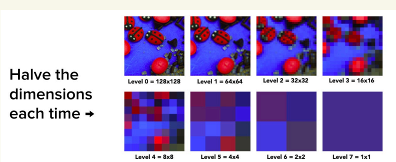


Mip-mapping:

1. pre-compute lower res versions of texture
2. store textures in mipmap
3. Adaptively choose mipmap level D according to scene

$$D = \log_2 L$$

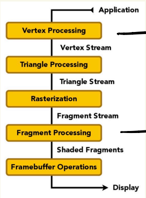
$$L = \max\left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right)$$



Bilinear = only use ONE mipmap level

Trilinear = LERP between TWO mipmap levels

* BIG jump in texture space = far away → high level (low res)
* SMALL jump in texture space = close up → low level (high res)



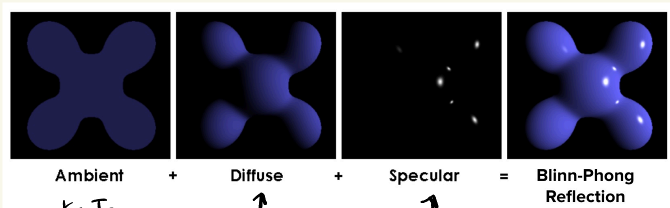
→ operations on geometry = transformations to screen space

→ operations on pixels = hidden surface removal / per-fragment shading

Reflection model

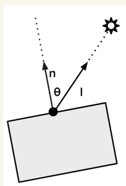
Blinn-Phong =

↑ default w/ Gouraud shading



$k_a I_a$

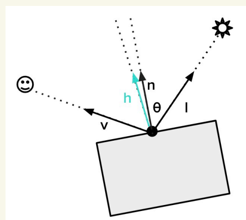
↑ DIFFUSE SHADING
$k_d \left(\frac{I}{r^2}\right) \max(0, n \cdot l)$



↑ SPECULAR SHADING

$h = \text{bisector}(v, l) = \frac{v + l}{\|v + l\|}$

$k_s \left(\frac{I}{r^2}\right) \max(0, n \cdot h)^p$ ← shininess



Phong shading = interpolate vertex normals per pixel
* slow

Gouraud shading = compute light per vertex
* FAST

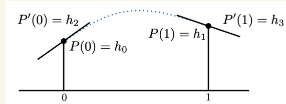
Hidden surface removal

: objects overlap → only display what's visible @ front

* track depth = z-value of fragments
  - ↳ pixel takes value of closest fragment in z-buffer (initialized to infinity)

Cubic Hermite interpolation : combine discrete pts into shape

input: values (p) & derivatives (p') @ endpoints
output: cubic polynomial that interpolates
soln: weighted sum of Hermite basis functions

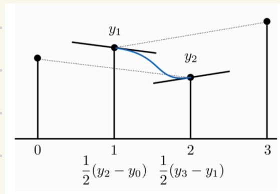


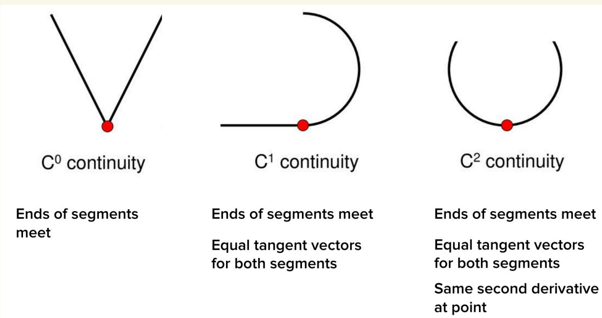
$P(t) = h_0 H_0(t) + h_1 H_1(t) + h_2 H_2(t) + h_3 H_3(t)$

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

Catmull Rom interpolation

input: sequence of points
① calculate slopes between alternating pts
② use Hermite interpolation
output: spline w/ C1 continuity





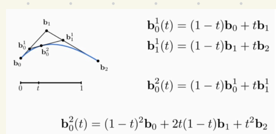
review math

Bezier curves

cubic Bezier: specify derivatives w/ control points

de Casteljau Algorithm = recursively compute intermediate control points through LERP

$P_i' = (1-t)P_i + t(P_{i+1})$



Catmull Rom = easy to use when you know set of control points ahead of time → ensures curve passes through ALL control pts (except maybe endpts)

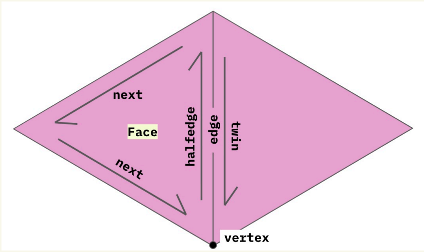
Halfedges = represent meshes to represent 3D shapes

```
struct Halfedge {
        Halfedge *twin,
        Halfedge *next;
        Vertex *vertex;
        Edge *edge;
        Face *face;
}

struct Vertex {
        Point pt;
        Halfedge *halfedge;
}

struct Edge {
        Halfedge *halfedge;
}

struct Face {
        Halfedge *halfedge;
}
```

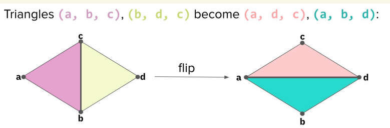


* useful to use do-while loops
* → next() to get next edge
  → twin() to get opposite halfedge

* Review coding examples!!! *

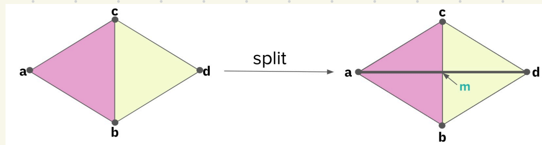
Edge flip = flip a halfedge

* no elements created or destroyed *



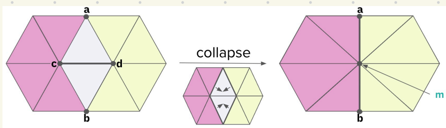
Edge split = insert midpoint into halfedge

* elements are added *



Edge collapse = replace edge (c, d) w/ vertex m

* delete elements * (lose 2 faces for △)

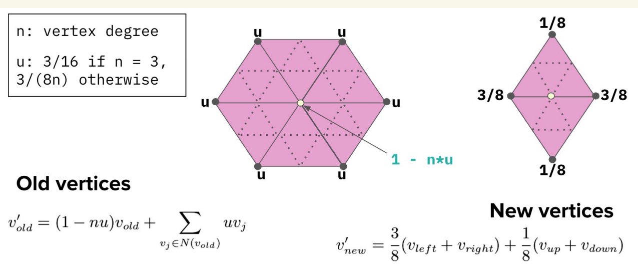


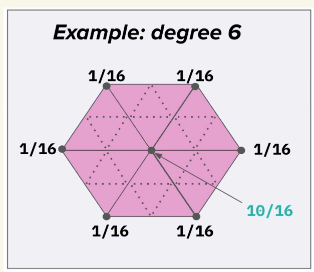
Subdivision : coarse mesh → smooth algorithmically

Loop subdivision = for triangle meshes

① split each △ face into 4 → new △, new vertices

② update old & new vertex positions as weighted sum





③ for any new edge that connects new to old vertex → EDGE FLIP

Catmull Clark Subdivision: meshes w/ variable polygon

1. Add vertex in each face
2. Add midpoint to each edge
3. Connect all new vertices
4. Adjust vertex positions to weighted avg

Ray Tracing = traces path of light as rays that travel through scene

1. Cast ray from camera into scene
2. Check whether ray intersects any objects in scene
3. Determine how intersecting surface should be shaded
4. If object reflective → generate NEW ray reflecting off surface & trace that ray
   - ↳ If object transparent: refraction occurs *Snell's law to bend light*
5. Shadow rays: rays cast from pt of intersection toward each light source
6. Global illumination: where light bounces off surfaces → indirect lighting

$r(t) = o + td$

- position of origin → o
- time → t
- direction of ray → d

Plane equation: $(p - p') \cdot N = 0$

- any pt on plane → p'
- normal vector → N

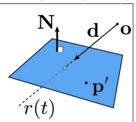
Ray-plane intersection:

Ray equation:

$$\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d}, \quad 0 \le t < \infty$$

Plane equation:

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0$$



Solve for intersection

Set $\mathbf{p} = \mathbf{r}(t)$ and solve for t

$$(\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = (\mathbf{o} + t\,\mathbf{d} - \mathbf{p}') \cdot \mathbf{N} = 0$$

$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}} \qquad \textbf{Check: } 0 \le t < \infty$$

$(p - p') \cdot N = 0$

$(o + td - p') \cdot N = 0$

$(o - p') \cdot N + td \cdot N = 0$

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

$t < 0$ → intersection BEHIND origin so INVALID intersection

$d \cdot N = 0$ → direction perpendicular to plane's normal vector = ray is PARALLEL

↳ $(o - p') \cdot N = 0$ → infinite intersections

$(o - p') \cdot N \neq 0$ → zero intersections

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

1. Set $f(o + td) = 0$ and solve for t
2. Plug values of t back into ray equation
3. ID where ray first hits surface

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

$$\mathbf{r}(t) = (0,0,0) + t\,(1,1,0)$$

$x = 0 + t \cdot 1 = t$

$y = 0 + t \cdot 1 = t$

$z = 0 + t \cdot 0 = 0$

sub back into here

$$\frac{(x-2)^2}{4} + (y-2)^2 + \frac{z^2}{4} - 1 = 0$$

$$\frac{(t-2)^2}{4} + (t-2)^2 - 1 = 0$$

$$\frac{5}{4}(t-2)^2 = 1$$

$$(t-2)^2 = \frac{4}{5} \quad \Rightarrow \quad t = 2 \pm \sqrt{\frac{4}{5}}$$

plug first t back into ray equation

$r(t) = (0,0,0) + \left(2 - \sqrt{\tfrac{4}{5}}\right)(1,1,0)$

$= \left(2 - \sqrt{\tfrac{4}{5}},\ 2 - \sqrt{\tfrac{4}{5}},\ 0\right)$

first intersection point

Ray-Triangle Intersection

$P = \alpha P_0 + \beta P_1 + \gamma P_2 \longrightarrow P = (1-b_1-b_2)P_0 + b_1 P_1 + b_2 P_2$

$= P_0 - b_1 P_0 - b_2 P_0 + b_1 P_1 + b_2 P_2$

Let $b_1 = \beta$, $b_2 = \gamma \rightarrow \alpha = 1 - b_1 - b_2$

point within triangle

↓ Plug in $P = O + tD$

$O + tD = P_0 + b_1(P_1 - P_0) + b_2(P_2 - P_0)$

$O - P_0 = tD + b_1(P_1 - P_0) + b_2(P_2 - P_0)$

$M \quad x \quad = \quad b$

$$\begin{bmatrix} -D & P_1 - P_0 & P_2 - P_0 \end{bmatrix} \begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = O - P_0$$

Möller-Trombore Algorithm

Möller-Trombore Algorithm = efficient way to determine if ray intersects △

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{S_1 \cdot E_1} \begin{bmatrix} S_2 \cdot E_2 \\ S_1 \cdot S \\ S_2 \cdot D \end{bmatrix}$$

$t \geq 0$

$0 \leq b_1 \leq 1$

$0 \leq b_2 \leq 1$

$0 \leq 1 - b_1 - b_2 \leq 1$

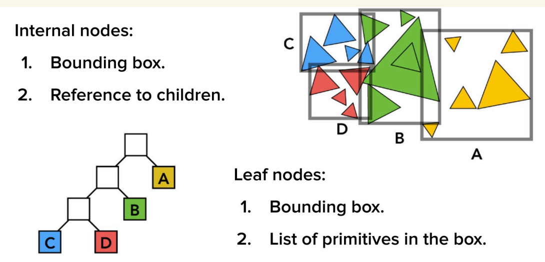
Acceleration

Bounding Volume Hierarchy

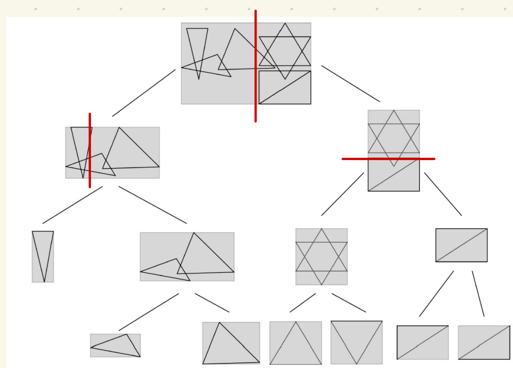
Bounding Volume Hierarchy = organizes objects into tree structure where each node has bounding volume which encapsulates set of objects/primitives

Goal: reduce # of objects that need to be checked for intersections

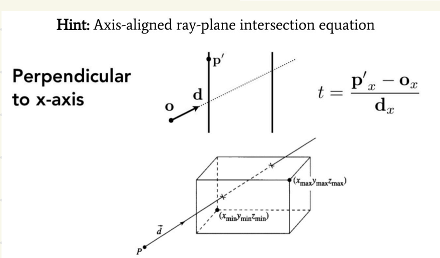
* if ray does NOT intersect boundry volume → skip checking individual objects within it



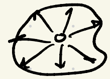
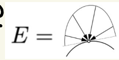
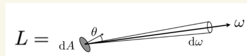
1. Always pick the longest axis to divide
2. Use center of mass to decide their relative pos
3. Keep BVH balanced → try to ensure same # of triangles for children nodes

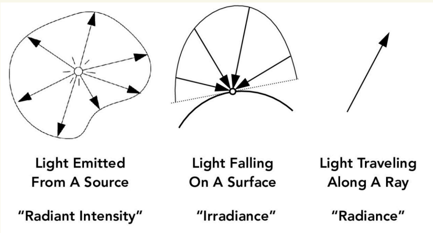


```
Intersect (Ray ray, BVH node)
  if (ray misses node.bbox) return;
  if (node is a leaf node)
     test intersection with all objs;
     return closest intersection;
  hit1 = Intersect (ray, node.child1);
  hit2 = Intersect (ray, node.child2);
  return closer of hit1, hit2;
```



Radiometry

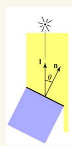
Flux (power)	watts (W)	$\Phi = \frac{dQ}{dt}$	total energy per time
Radiant intensity	flux / solid angle (W / sr)	$I(\omega) = \frac{d\Phi}{d\omega}$ 	amt of radiant flux in specific direction per solid angle (think flashlight)
Irradiance	flux / area / solid angle (W / m²)	$E(x) = \frac{d\Phi(x)}{dA} = \frac{\Phi}{A}\cos\theta$ 	how much power is received by a surface (sunlight hitting solar panel)
Radiance	flux / area (W / sr m²)	$L(p,\omega) = \frac{d^2\Phi(p,\omega)}{d\omega\, dA\cos\theta}$ 	how much light is traveling from in a specific direction from a surface (computer screen from diff angles)



Symbol/Name	Radiometry Unit/Name	Photometry Unit/Name	Effect of Increased R
Q: Energy	Radiant Energy Joules (W·s)	Luminous Energy Lumen·sec	=
Φ: Flux (Power)	Radiant Power W	Luminous Power Candela·sr	=
I: Angular Flux Density	Radiant Intensity W/sr	Luminous Intensity Candela = Lumen/sr	=
E: Spatial Flux Density	Irradiance (in), Radiosity (out) W/m²	Illuminance (in), Luminosity (out) Lux = Lumen/m²	↓
L: Spatio-Angular Flux Density	Radiance W/m²/sr	Luminance Nit = Candela/m²	=

Physics Symbol/-Name	Radiometry Unit/Name	Photometry Unit/Name	Definition
Q Energy	Radiant Energy **Joules (W·s)**	Luminous Energy **Lumen·sec**	$Q = \int_{t_0}^{t_1} \Phi dt$
Φ Flux(Power)	Radiant Power **W**	Luminous Power **Lumen (Candela·sr)**	$\Phi = \frac{dQ}{dt}$
I Angular Flux Density	Radiant Intensity **W/sr**	Luminous Intensity **Candela (Lumen/sr)**	$I(\omega) = \frac{d\Phi}{d\omega}$
E Spatial Flux Density	Irradiance (in), Radiosity (out) **W/m²**	Illuminance (in), Luminosity (out) **Lux (Lumen/m²)**	$E(p) = \frac{d\Phi(p)}{dA}$
L Spatio-Angular Flux Density	Radiance **W/m²/sr**	Luminance **Nit (Candela/m²)**	$L(p,\omega) = \frac{d^2\Phi(p,\omega)}{d\omega dA\cos\theta} = \frac{dE(p)}{d\omega\cos\theta} = \frac{dI(p,\omega)}{dA\cos\theta}$

Lambert's law

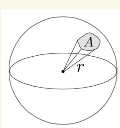


power / unit area proportional to $\cos\theta = l \cdot n \rightarrow E = \frac{\Phi}{A}\cos\theta$

solid angle = measure of how large object appears from given point in 3D space

↳ ratio of subtended area on sphere to radius squared

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



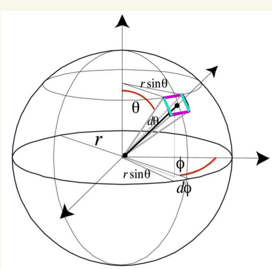
sphere = 4π steradians

hemisphere = 2π steradians

Regular angle = ratio of subtended arc length on circle to radius

$$\theta = \frac{l}{r}$$

circle = 2π radians



$$dA = (r\, d\theta)(r\sin\theta\, d\phi) = r^2 \sin\theta\, d\theta\, d\phi$$

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

Probability

PMF = $P[X=x]$

Expectation: $E[X] = \sum_i x_i p_i = \int x p(x) dx$

CDF = $F(x) = \int_{-\infty}^{x} p(t) dt$

Inversion Method:
1. Calculate CDF: $F(x) = \int_{-\infty}^{x} p(t) dt$
2. invert CDF: $F^{-1}(x)$ → set U = CDF and solve for x (becomes inverse)
3. sampling X according to $p(x)$ achieved by sampling $U \rightarrow X = F^{-1}(U)$

Monte Carlo Integration

$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \longrightarrow E[F_N] = \int_a^b f(x) dx$$

* good to approximate complex shapes
* don't need a lot of samples

for importance sampling → need to map to $[0,1]$ $(U \rightarrow X)$

$x_i = -\frac{\pi}{2} \rightarrow u_i = 0$

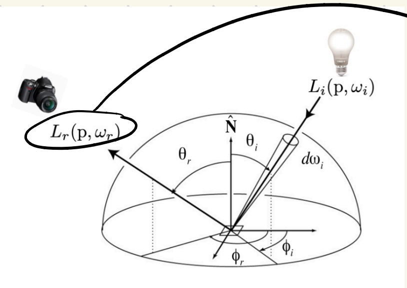
$x_i = \frac{\pi}{2} \rightarrow u_i = 1$

$$u_i = \frac{x_i}{\pi} + \frac{1}{2}$$

Light Transport

BRDF = bidirectional reflectance distribution function

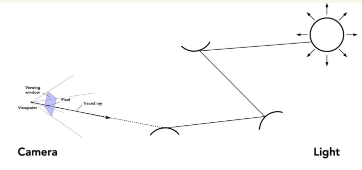
* ratio of reflected radiance in given outgoing direction w_o to incoming irradiance from dir w_i



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

Global Illumination

= recursively calculate how much light falls onto point p



1 bounce: radiance from light source → p → camera

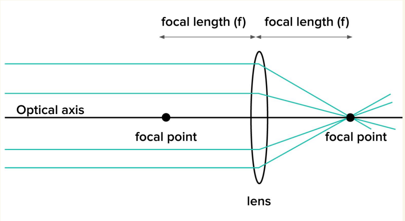
2 bounce: radiance from light source → p' → p → camera

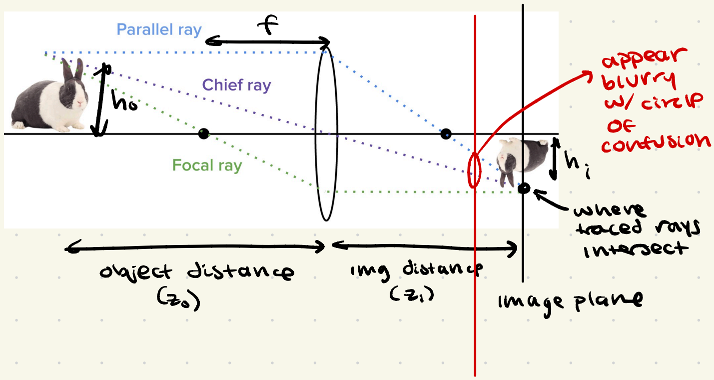
Termination conditions

* Russian Roulette: at each bounce, randomly terminate current ray w/ $p(x) = 1 - p_{rr}$

Importance sampling = sampling MORE from important areas → reduces variance

Cameras & lenses





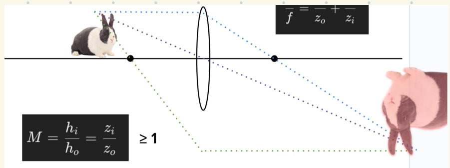
Thin lens Equation:

$$\frac{1}{f} = \frac{1}{z_o} + \frac{1}{z_i}$$

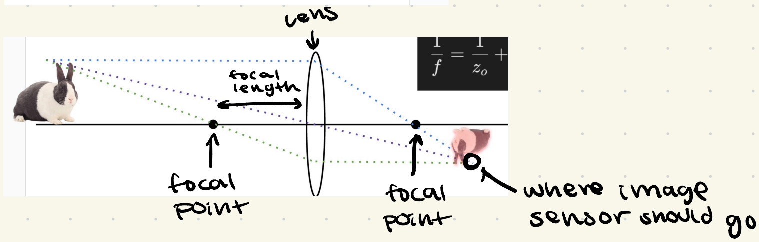
Magnification

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

≥ 1

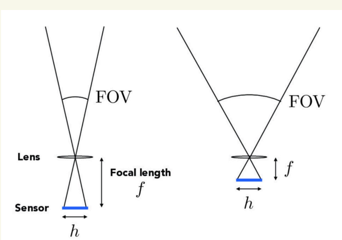


≤ 1



chief ray = pass through CENTER of lens
focal ray = emerges parallel to optical axis
parallel ray = directed through focal point on other side
focal point = where img is formed by the lens

FOV = angle of scene captured by camera lens



smaller focal length → LARGER FOV
larger focal length → SMALLER FOV

smaller sensor size → smaller FOV

$$FOV = 2\arctan\left(\frac{h}{2f}\right)$$

h = sensor height
f = focal length

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

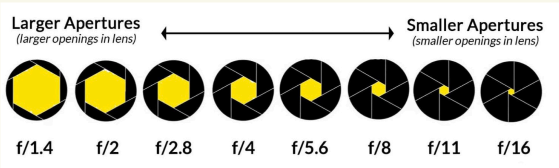
circle of confusion = optical spot caused by cone of light not coming to focus

smaller aperture → more concentrated rays → MORE depth of field (less blur)
= BIGGER f stop

large aperture → shallow depth of field

smaller f-stop → shallower depth of field

Exposure = irradiance * time * gain



larger aperture = higher irradiance

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

$\sqrt{2}$ increase in aperture DOUBLES light

x2 shutter duration → x2 exposure

x2 ISO gain → x2 exposure

shutter duration = seconds sensor exposed to light

ISO = sensor's sensitivity to light = how much light amplified

higher ISO → more noise

Simulation

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

x^t = position

$\dot{x}^t$ = velocity

$\ddot{x}^t$ = acceleration

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

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

✓ simple & easy to compute

✗ errors accumulate
↳ small step size to have low approximation error

Implicit Euler's Method / Backward Euler's

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

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

more stable

non-linear equation = DIFFICULT

Modified Euler's

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

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

more stable

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

Hooke's Law: $f_{a \to b} = k_s \frac{b-a}{\|b-a\|} (\|b-a\| - \ell)$

↑ spring constant (k_s)   ↑ length (ℓ)

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

$F = ma$

Animation

Forward kinematics = angles for joints → computer determines final position

Inverse kinematics = ending position → compute joint angles to reach position

- × no realistic soln
- × multiple possible solns → unique depending on how θ constrained

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

* usually interpolate between them

linear interpolation ≠ rotations

- linear interpolation ↑ straight line
- rotations ↑ circle
- ≠ ↑ not good match for

Light

Spectral power distribution = non-negative function giving power in light beam @ given wavelength

* characterizes light source
* watts/nm
* ADDITIVE!

Monospectral Distribution = single color

$R \cdot s_R(\lambda) + G \cdot s_G(\lambda) + B \cdot s_B(\lambda) = \begin{bmatrix} | & | & | \\ s_R & s_G & s_B \\ | & | & | \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = s_{disp}(\lambda)$

- R ↑ red brightness [0,1]
- $s_R(\lambda)$ ↑ red monospectrum

cone cells = diff sensitivities

L	= orange/yellow	=	$\int r_L(\lambda) s(\lambda) d\lambda$
M	= green/yellow	=	$\int r_M(\lambda) s(\lambda) d\lambda$
S	= blue	=	$\int r_L(\lambda) s(\lambda) d\lambda$

↑ response curve (r)   ↑ SPD (s)

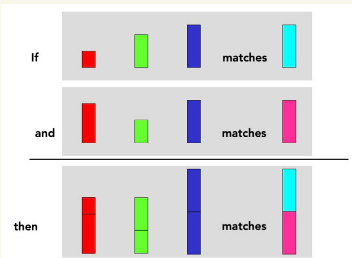
$\begin{bmatrix} S \\ M \\ L \end{bmatrix} = \underbrace{\begin{bmatrix} — r_S — \\ — r_M — \\ — r_L — \end{bmatrix}}_{R} \underbrace{\begin{bmatrix} | \\ s \\ | \end{bmatrix}}_{S}$

metamer = 2 DIFF spectra w/ same visual (S, M, L) response
* reproduce real-world scenes w/ HARD to recreate spectra

$$\underbrace{\begin{bmatrix} — & r_S & — \\ — & r_M & — \\ — & r_L & — \end{bmatrix} \begin{bmatrix} | & | & | \\ s_R & s_G & s_B \\ | & | & | \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}}_{\text{color we perceive on display}} = \underbrace{\begin{bmatrix} — & r_S & — \\ — & r_M & — \\ — & r_L & — \end{bmatrix} \begin{bmatrix} | \\ s \\ | \end{bmatrix}}_{\text{color we perceive in real life}}$$

Gamut = range of colors that can be displayed on a device **[* device dependent *]**

* color matching w/ primary lights
  - → not possible = add NEG amt of color into test side
* color matching = LINEAR



3 primary colors = necessary for normal color vision

2 primary colors = red green colorblindness

Human Eye

Rods = primarily in low light, diff shades of gray

cones = "photopic" receptors → sensation of color

Hue, Saturation, Value (HSV)

Hue = dominant wavelength (what color)

Saturation = how vivid the color is

value = amount of light

chromaticity digram

* pure saturated spectral at corners, desaturated in center
* does NOT include black

CIELAB = strives for perceptual uniformity

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} r_S \cdot s_R & r_S \cdot s_G & r_S \cdot s_B \\ r_M \cdot s_R & r_M \cdot s_G & r_M \cdot s_B \\ r_L \cdot s_R & r_L \cdot s_G & r_L \cdot s_B \end{bmatrix}^{-1} \begin{bmatrix} — & r_S & — \\ — & r_M & — \\ — & r_L & — \end{bmatrix} \begin{bmatrix} | \\ s \\ | \end{bmatrix}$$

negative color values = gamut not enough to display soln

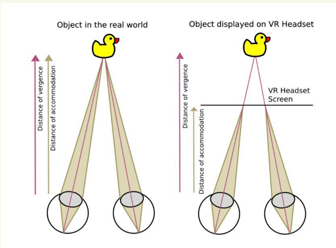
Virtual Reality

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

accomodation = eye's ability to change shape of lens to bring objects into focus

* lens accomodate physical screen distance, <u>NOT</u> virtual depth

vergence - accomodation conflict = vergence cues of virtual distance
accomodation cues fixed at display distance
MISMATCH SIGNALS = visual discomfort

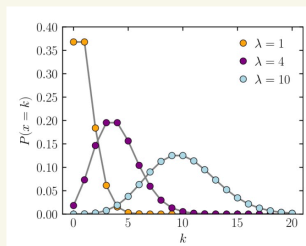


* low-latency tracking/rendering
* monoscopic = both eyes w/ same 360° frame → no stereoscopic depth
* MORE FOV = enhanced realism → can reduce angular resolution
* real + virtual objects = mixed reality

sensors

* photons enter according to Poisson distribution
  - ↳ occurrence 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

signal has more variability → Poisson shot noise