Transformation

Homogeneous Coordinate

3D Point: $(x, y, z, 1)^\intercal$
3D Vector: $(x, y, z, 0)^\intercal$
Vector ± Vector = Vector
Point ± Vector = Point
Point – Point = Vector

Point + Point = ? $\quad (x, y, z, w)^\intercal = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1\right)^\intercal$

Basic Linear Transformation

Translation: $\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & & & t_x \\ & 1 & & t_y \\ & & 1 & t_z \\ & & & 1 \end{pmatrix}$

Rotation: $\mathbf{R_x}(\theta) = \begin{pmatrix} 1 & & & \\ & \cos\theta & -\sin\theta & \\ & \sin\theta & \cos\theta & \\ & & & 1 \end{pmatrix} \qquad \mathbf{R}(-\theta) = \mathbf{R}(\theta)^{-1} = \mathbf{R}(\theta)^\intercal$

Scaling: $\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & & & \\ & s_y & & \\ & & \mathrm{s_z} & \\ & & & 1 \end{pmatrix}$

Shear: $\begin{pmatrix} 1 & a & & \\ & 1 & & \\ & & 1 & \end{pmatrix}$

Affine Transformation

$$\begin{pmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ & & & 1 \end{pmatrix}$$

Euler Angle

Z (Roll) → X (Pitch) → Y (Yaw)
Gimbal Lock: mapping of Euler angles and rotations is n to 1
Quaternion: (x, y, z, w) used for linear interpolation

Rodrigues' Rotation Formula

Rotation by angle α around axis $\mathbf{n}$

$$\mathbf{R}(\mathbf{n},\alpha) = \cos(\alpha)\mathbf{I} + (1-\cos(\alpha))\mathbf{n}\mathbf{n}^{\mathsf{T}} + \sin(\alpha)\begin{pmatrix} & -n_z & n_y \\ n_z & & -n_x \\ -n_y & n_x & \end{pmatrix}$$

Viewing Transformation

Transform camera to its original transformation, and the world moves correspondingly to keeps the same relative transformation

Position: $\mathbf{t} = (t_x, t_y, t_z)$ → $(0,0,0)$

Forward: $\mathbf{f} = (f_x, f_y, f_z)$ → -Z Axis

Upward: $\mathbf{u} = (u_x, u_y, u_z)$ → Y Axis

- Translate camera to origin

$$\mathbf{T}_{\text{view}} = \begin{pmatrix} 1 & & & -t_x \\ & 1 & & -t_y \\ & & 1 & -t_z \\ & & & 1 \end{pmatrix}$$

- Get inversed matrix of rotation

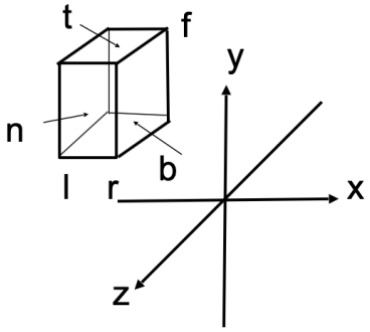
$$\mathbf{R}_{\text{view}}^{-1} = \begin{pmatrix} (\mathbf{f}\times\mathbf{u})_x & u_x & -f_x & \\ (\mathbf{f}\times\mathbf{u})_y & u_y & -f_y & \\ (\mathbf{f}\times\mathbf{u})_z & u_z & -f_z & \\ & & & 1 \end{pmatrix}$$

- Get viewing transformation matrix

$$\mathbf{M}_{\text{view}} = \mathbf{R}_{\text{view}}\mathbf{T}_{\text{view}} = \left(\mathbf{R}_{\text{view}}^{-1}\right)^{\mathsf{T}}\mathbf{T}_{\text{view}}$$

Orthographic Projection

Camera goes to origin, drop Z coordinate, translate and scale image to $[-1,1]^2$



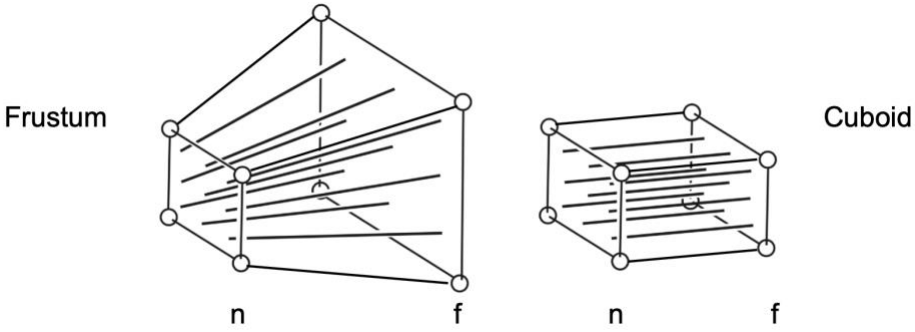
Generally, map the cuboid $[l,r]\times[b,t]\times[f,n]$ to a canonical cube $[-1,1]^3$

$$\mathbf{M}_{\text{ortho}} = \mathbf{S}_{\text{ortho}}\mathbf{T}_{\text{ortho}} = \begin{pmatrix} \frac{2}{r-l} & & & \\ & \frac{2}{t-b} & & \\ & & \frac{2}{n-f} & \\ & & & 1 \end{pmatrix}\begin{pmatrix} 1 & & & -\frac{r+l}{2} \\ & 1 & & -\frac{t+b}{2} \\ & & 1 & -\frac{n+f}{2} \\ & & & 1 \end{pmatrix}$$

Field-of-View (FOV): sometimes prefer vertical FOV (fovY)
Aspect ratio = width / height

Perspective Projection

Transform the frustum (perspective projection) to the cuboid (orthographic projection)



$$\mathbf{M}_{\text{persp}} = \mathbf{M}_{\text{ortho}}\mathbf{M}_{\text{persp}\to\text{ortho}}$$

$$\mathbf{M}_{\text{persp}}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx/z \\ ny/z \\ \text{unknown} \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix}$$

By consider immobile points $(0,0,n)$ and $(0,0,f)$, get unknown coefficients

$$\mathbf{M}_{\text{persp}\to\text{ortho}} = \begin{pmatrix} n & & & \\ & n & & \\ & & n+f & -nf \\ & & 1 & \end{pmatrix}$$

Rasterization

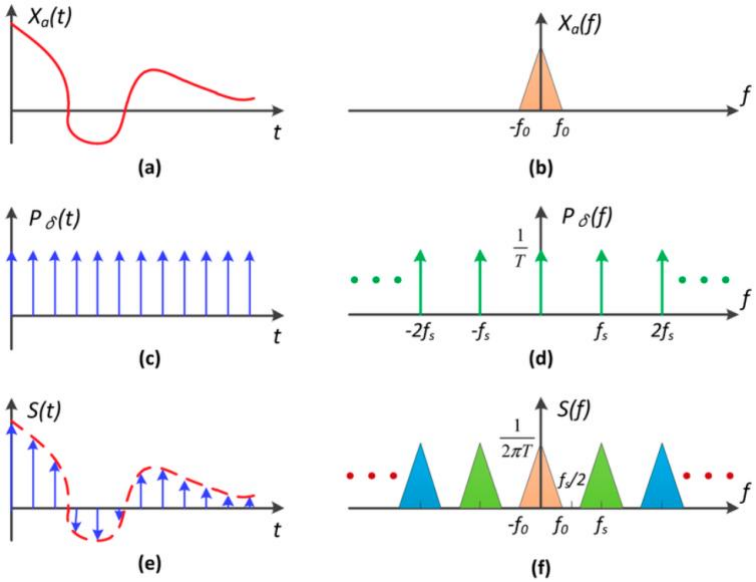
Break up polygons into triangles, and draw projected triangles on screen and set pixel values

Convolution Theorem

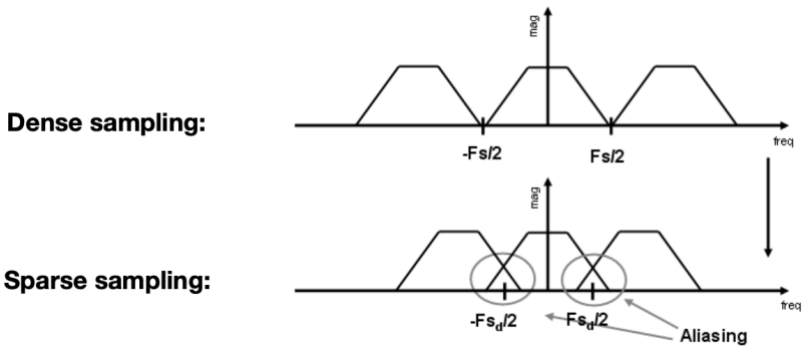
Convolution in spatial domain = Multiplication in frequency domain
Convolution in frequency domain = Multiplication in spatial domain

[Basic] Sampling

Sampling = repeat frequency contents

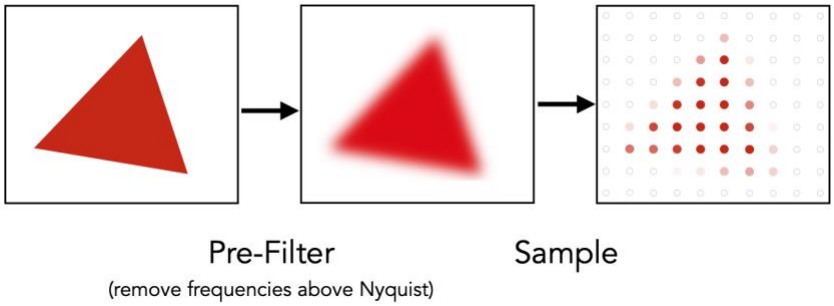


Aliasing = mixed frequency contents



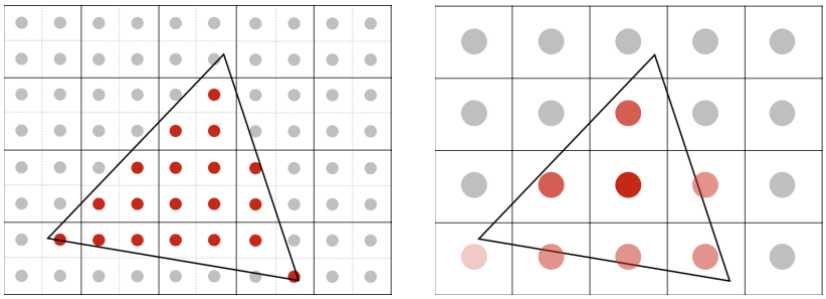
Anti-Aliasing

- Increase sampling rate
- Blurring or Pre-Filtering (filtering our high frequencies) before sampling



Super-Sampling (MSAA)

Sampling multiple location within a pixel and averaging their values



Other milestones:

- FXAA (Fast Approximate AA)
- TAA (Temporal AA)
- DLSS (Deep Learning Super-Sampling)

Z-Buffer

Store current minimum Z-value for each sample (pixel)

(1) Frame buffer stores color values

(2) Depth buffer (Z-buffer) stores depth

Shadow Mapping

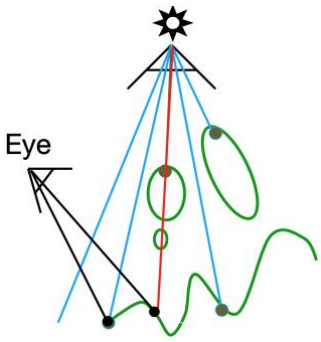
Two passes:

(1) Pass 1: get depth image from light source (shadow map, Z-buffer of light source)

(2) Pass 2A: standard image (with depth) from eye

(3) Pass 2B: project visible points in eye view back to light source

(4) Compare distance from light source of each point with the depth of corresponding pixel in shadow map: if not match, the point is in shadow

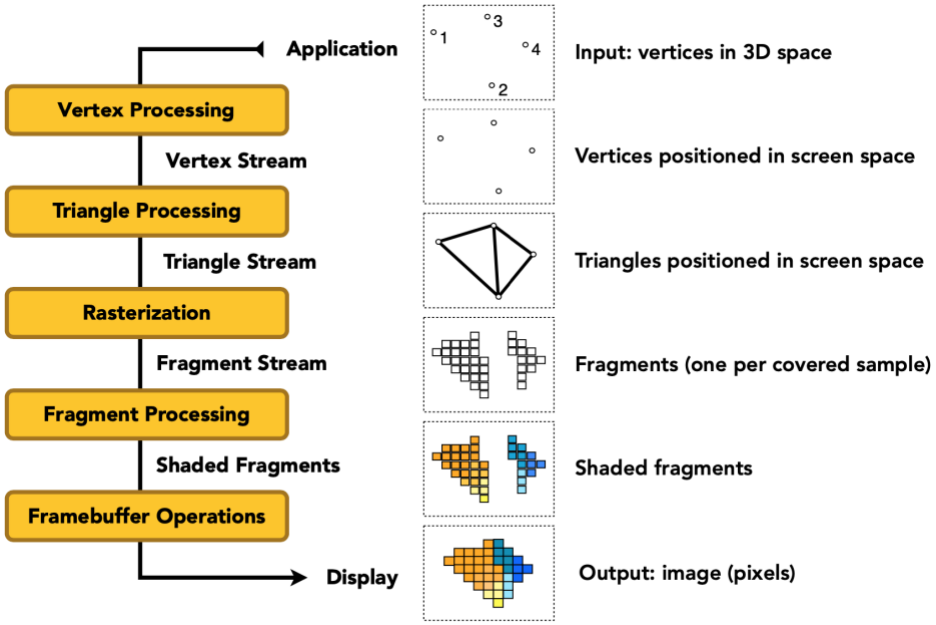


Problems:

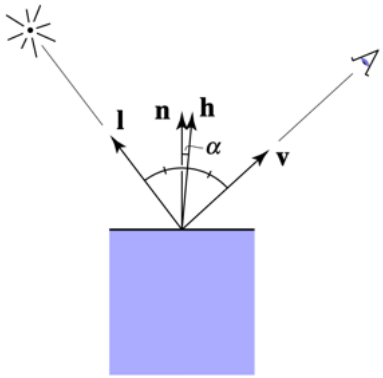
- Hard shadows (point light only)
- Quality depends on shadow map resolution
- Involves equality comparison of floating point depth values means issues of scale, bias, tolerance

Shading

Graphics (Real-time Rendering) Pipeline



Blinn-Phong Reflectance Model



$$L_{\text{diffuse}} = k_{\text{diffuse}} \frac{I}{r^2} \max(0, \mathbf{n} \cdot \mathbf{l})$$

$$L_{\text{specular}} = k_{\text{specular}} \frac{I}{r^2} \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

$$L_{\text{ambient}} = k_{\text{ambient}} I_{\text{ambient}}$$

$$L = L_{\text{diffuse}} + L_{\text{specular}} + L_{\text{ambient}}$$

L: reflected light

k, p: coefficient

$\frac{I}{r^2}$: energy arrived at the shading point (intensity / (distance from the light source)^2)

$\mathbf{h} = \frac{\mathbf{v}+\mathbf{l}}{\|\mathbf{v}+\mathbf{l}\|}$: bisector of $\mathbf{v}$ and $\mathbf{l}$

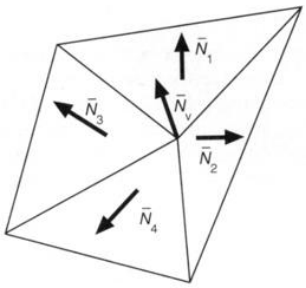
Shading Models

Flat shading: shade each triangle
Gouraud shading: shade each vertex
Phong shading: shade each fragment (e.g., pixel)

Interpolation of Vertex Normal Vector

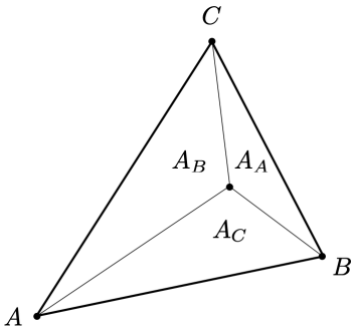
Average surrounding faces' normal vectors

$$N_v = \frac{\sum_i N_i}{\|\sum_i N_i\|}$$



Barycentric Coordinate

A coordinate system for triangles (α, β, γ)



$$P = \alpha A + \beta B + \gamma C$$

$$\alpha + \beta + \gamma = 1, \ \alpha \geq 0, \ \beta \geq 0, \ \gamma \geq 0$$

Only work triangles in 3D space, not equals to work for 3D triangles in a 2D raster screen

$$\alpha = \frac{A_A}{A_A + A_B + A_C} = \frac{(P - B) \times (C - B)}{(A - B) \times (C - B)}$$

$$\beta = \frac{A_B}{A_A + A_B + A_C} = \frac{(P - C) \times (A - C)}{(B - C) \times (A - C)}$$
$$\gamma = \frac{A_C}{A_A + A_B + A_C} = 1 - \alpha - \beta$$

Texture

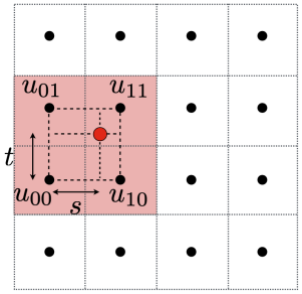
Get texture coordinate (u, v) of a point, and sample (u, v) on the texture

Texture Magnification Problems

Texel: a pixel on a texture

- Jaggies (Oversampling): Texture is too small
  - Bilinear Interpolation
  - Bicubic Interpolation
- Moiré Pattern (Undersampling): A pixel covers a range of texels
  - Mipmap
  - Anisotropic Filtering

Bilinear Interpolation

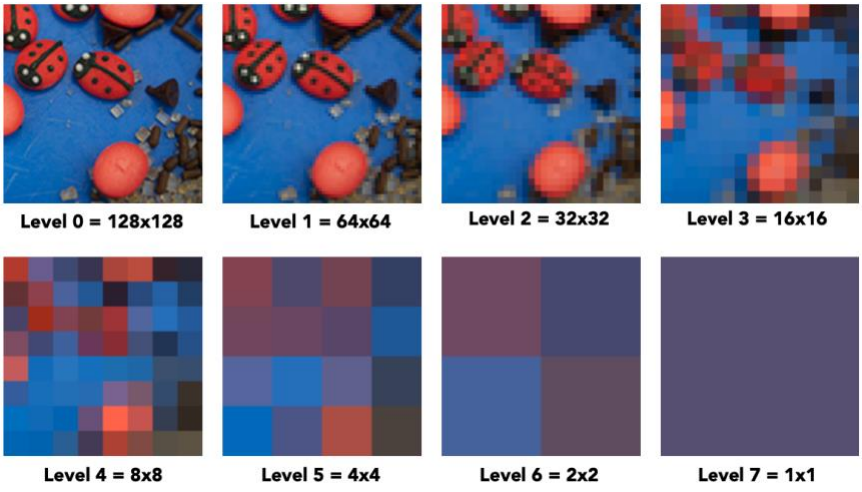


$$\text{Lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$
$$f(x, y) = \text{Lerp}\big(s, \text{Lerp}(t, u_{00}, u_{01}), \text{Lerp}(t, u_{10}, u_{11})\big)$$

MipMap

Allowing fast, approximate and square range queries

- Additional storage: 1/3
- Limitation: overblur



- Pre-store a hierarchy of different levels of raw images
- Estimate texture coordinates (u, v) of neighboring screen samples a_i
- Computing mipmap level by the size of required range

$$D = \log_2\left(\max_{a_i}\sqrt{\left(\frac{\mathrm{d}u}{\mathrm{d}a_i}\right)^2 + \left(\frac{\mathrm{d}v}{\mathrm{d}a_i}\right)^2}\right)$$

- Query the texel in level D by bilinear interpolation
  Trilinear Interpolation: Linear interpolation based on bilinear results of level D and $D+1$

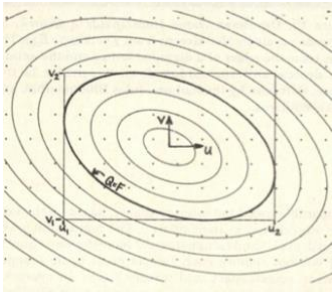
Anisotropic Filtering

Ripmap: axis-aligned rectangle zones

- Additional storage: 3x
- Limitation: diagonal zones

EWA Filtering

Multiple lookups, weighted average



Environment Mapping

Map environment lighting into a texture (light from different directions)

- Spherical map
- Cube map (6 faces): bounding box of a sphere; the light from the sphere hit the cube face

Bump/Normal Mapping

Fake the detailed geometry without adding more triangles

Perturb surface normal per pixel (for shading computation only) by height shift per texel defined by a texture

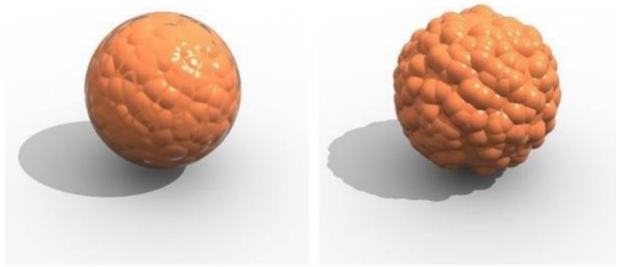
(1) Original surface normal $\mathbf{n_p} = (0, 0, 1)$

(2) Derivatives at $\mathbf{p}$ are

$$\frac{\mathrm{d}\,\mathbf{p}}{\mathrm{d}\,u} = c_1\big(h(u+1) - h(u)\big)$$

$$\frac{\mathrm{d}\,\mathbf{p}}{\mathrm{d}\,v} = c_1\big(h(v+1) - h(v)\big)$$

(3) Perturbed normal is $\mathbf{n} = \left(-\frac{\mathrm{d}\,\mathbf{p}}{\mathrm{d}\,u}, -\frac{\mathrm{d}\,\mathbf{p}}{\mathrm{d}\,v}, 1\right)$. normalized



Bump / **Normal** mapping Displacement mapping

Displacement Mapping

Use the same texture as in bump mapping
Actually moves the heights of vertices (need triangles to be small enough)

3D Procedural Noise + Solid Modeling

<Procedural Appearance>
3D texture generated by 3D function
e.g., Perlin noise

Ambient Occlusion Texture Map

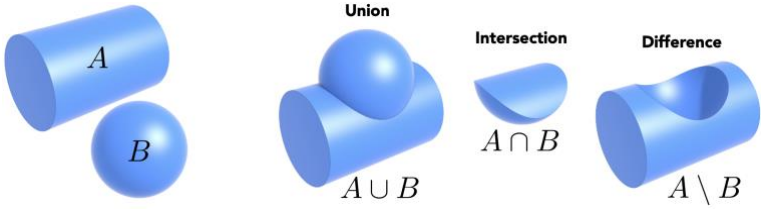
Precomputed shadow map (0-1), then multiply simple shading to generate ambient occlusion

Geometry

Implicit Representation of Geometry

Based on classify points – points satisfy some specified relationship
* Hard to get all points on the surface
* Easy to test points inside/outside/distance the surface

- Algebraic Surface: e.g., for point (x, y, z) where $f(x, y, z) = 0$
- Constructive Solid Geometry: combine implicit geometry via Boolean operations



- Signed Distance Function: given minimum distance (+/-) from anywhere to object; good for blending
- Fractals: exhibit self-similarity, detail at all scales

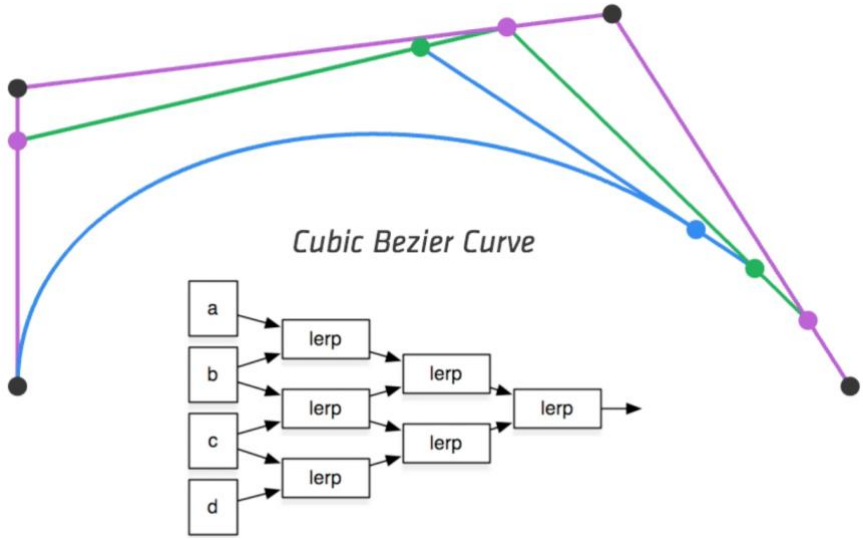
Explicit Representation of Geometry

All points are given directly or via parameter mapping

* Easy to get all points on the surface
* Hard to test points inside/outside/distance the surface

- Point Cloud: list of points
- Polygon Mesh: store vertexes & polygons (often triangles or quads)
- Parameter Mapping: e.g., given points (u, v) and $f: \mathbb{R}^2 \to \mathbb{R}^3$, can enumerate all points $(u, v) \mapsto (x, y, z)$

Bézier Curve



de Casteljau Algorithm

Recursively lerp control points

$$\mathbf{b}^n(t) = \sum_{j=0}^{n} \mathbf{b}_j B_j^n(t)$$

$\mathbf{b}^n(t)$: Bézier curve of order n
$\mathbf{b}_j$: Bézier control points

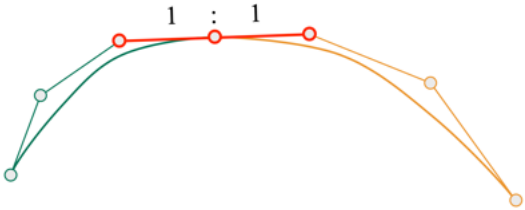
$B_j^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$: Bernstein polynomials

Properties of Cubic Bézier

- Interpolates endpoints: $\mathbf{b}(0) = \mathbf{b}_0,\ \mathbf{b}(1) = \mathbf{b}_3$
- Tangent to end segments: $\mathbf{b}'(0) = 3(\mathbf{b}_1 - \mathbf{b}_0), \mathbf{b}'(1) = 3(\mathbf{b}_3 - \mathbf{b}_2)$
- Affine transformation remains the same
- Curve is within convex hull of control points

Piecewise Bézier Curves

Higher-order Bézier curve not remain details, instead, chain many low-order Bézier curves
Piecewise cubic Bézier: the most common technique



C^0 and C^1 continuity: $\mathbf{b}_{0,3} = \mathbf{b}_{1,0} = \frac{1}{2}(\mathbf{b}_{0,2} + \mathbf{b}_{1,1})$

Splines

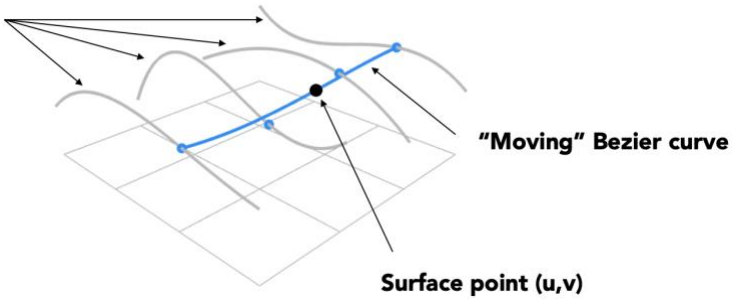
A continuous curve constructed so as to pass through a given set of points and have a certain number of continuous derivatives

B-splines: basis splines

NURBS

Bézier Surface

Bicubic Bézier Surface Patch: 4×4



(u,v)-separable application of de Casteljau algorithm

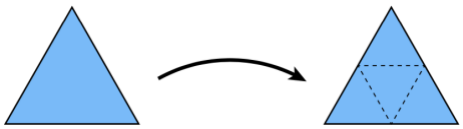
(1) Use de Casteljau to evaluate point u on each of the 4 Bezier curves. This gives 4 secondary control points

(2) Use de Casteljau to evaluate point v on the "moving" curve

Loop Mesh Subdivision

Common subdivision rule for triangle meshes

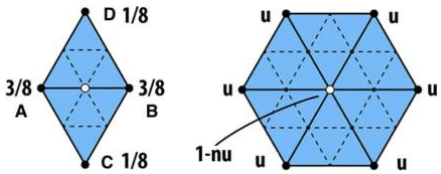
(1) Split each triangle into four



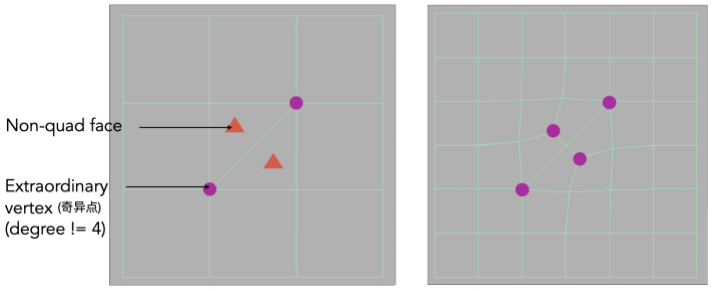
(2) Update positions of vertices

- For new vertices: $\mathbf{P} = \frac{3}{8}(\mathbf{A} + \mathbf{B}) + \frac{1}{8}(\mathbf{C} + \mathbf{D})$
- For old vertices: $\mathbf{P} = (\mathbf{1} - \deg(\mathbf{P}) * u)\mathbf{P} + u * \sum \text{neighbor}(\mathbf{P})$

  $u = \frac{3}{16}$ if $\deg(\mathbf{P}) = 3$; $\frac{3}{8\deg(\mathbf{P})}$ otherwise



Catmull-Clark Mesh Subdivision



Subdivision for general mesh

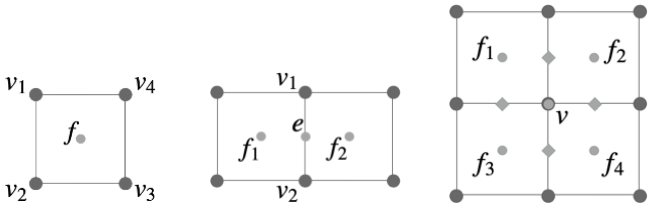
(1) Add vertex in each face, add midpoint on each edge, connect all new vertices
For each non-quad face, it will be subdivided into quad faces

(2) Update positions of vertices

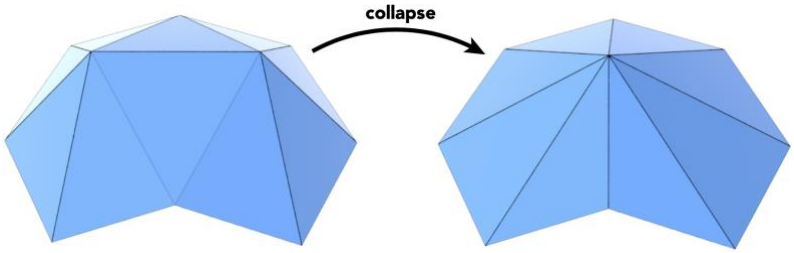
For face point: $\mathbf{f} = \frac{\mathbf{v}_1+\mathbf{v}_2+\mathbf{v}_3+\mathbf{v}_4}{4}$

For edge point: $\mathbf{e} = \frac{\mathbf{v}_1+\mathbf{v}_2+\mathbf{f}_1+\mathbf{f}_2}{4}$

For vertex point (old): $\mathbf{v} = \frac{\mathbf{f}_1+\mathbf{f}_2+\mathbf{f}_3+\mathbf{f}_4+2(\mathbf{e}_1+\mathbf{e}_2+\mathbf{e}_3+\mathbf{e}_4)+4\mathbf{v}}{16}$



Collapsing Edges Mesh Simplification



Quadric Error Metric: New vertex should minimize its sum of square distance (L2 distance) to previously related triangle planes

Iteratively collapse edges with greedy algorithm:

(1) Approximate distance to surface as sum of distances to planes containing triangles

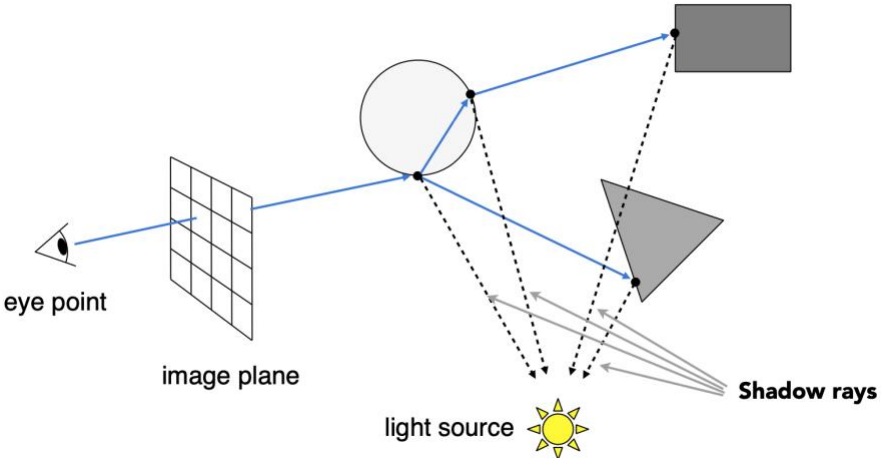
(2) Collapse edge with smallest score

(3) Maintain priority queue

Ray Tracing

Rasterization can only handle rays with no more than one reflection; ray tracing is for more realistic rendering

Whitted-Style (Recursive) Ray Tracing



- Always perform specular reflections / refractions
- Stop bouncing at diffuse surface

Ray Intersection with Plane

Ray Equation: $\mathbf{r}(t) = \mathbf{O} + t\mathbf{D}$
Plane Equation: $\mathbf{p}: (\mathbf{p} - \mathbf{P'}) \cdot \mathbf{N} = \mathbf{0}$, where $\mathbf{P'}$ is any point on the plane

$$t = \frac{(\mathbf{P'} - \mathbf{O}) \cdot \mathbf{N}}{\mathbf{D} \cdot \mathbf{N}}$$

Möller Trumbore Algorithm

Fast approach, giving barycentric coordinate directly

$$\mathbf{O} + t\mathbf{D} = (1 - b_1 - b_2)\mathbf{P}_0 + b_1\mathbf{P}_1 + b_2\mathbf{P}_2$$

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\mathbf{S}_1 \cdot \mathbf{E}_1} \begin{bmatrix} \mathbf{S}_2 \cdot \mathbf{E}_2 \\ \mathbf{S}_1 \cdot \mathbf{S} \\ \mathbf{S}_2 \cdot \mathbf{D} \end{bmatrix}$$

Where $\mathbf{E}_i = \mathbf{P}_i - \mathbf{P}_0$, $\mathbf{S} = \mathbf{O} - \mathbf{P}_0$, $\mathbf{S}_1 = \mathbf{D} \times \mathbf{E}_2$, $\mathbf{S}_2 = \mathbf{S} \times \mathbf{E}_1$
Check ray intersects triangle: $t, b_1, b_2, 1 - b_1 - b_2 > 0$

Axis-Aligned Bounding Box (AABB)

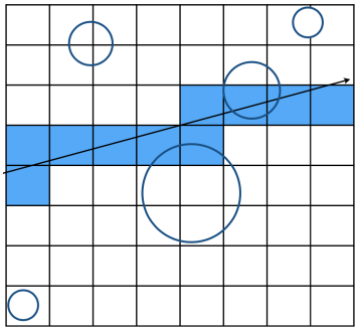
Any side of the bounding box is along either x, y, z axis
Box is the intersection of 3 pairs of slabs (left/right + front/back + top/bottom)

Slab intersection: $t = \frac{\mathbf{P}'_x - \mathbf{O}_x}{\mathbf{D}_x}$

Ray-AABB intersection:
(1) Computer intersections with slabs: $t_{\min}$ and $t_{\max}$
(2) For 3D box, $t_{\text{enter}} = \max\{t_{\min}\}$, $t_{\text{exit}} = \min\{t_{\max}\}$
(3) Ray and AABB intersect if and only if $t_{\text{enter}} < t_{\text{exit}}$ and $t_{\text{exit}} \geq 0$

Uniform Spatial Partition (Grid)

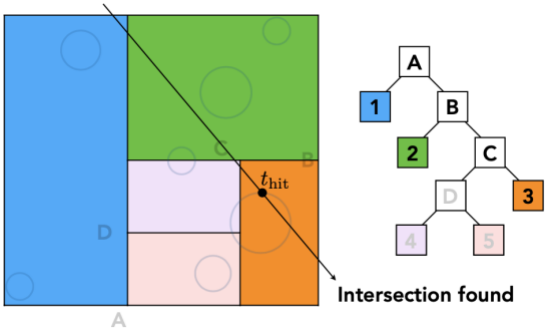
(1) Find bounding box, create grid
#cells = C * #objects; C≈27 in 3D
(2) Store each object in overlapping cells
(3) Step through grid in ray traversal order



KD-Tree (Spatial Partition)

(1) Find bounding box
(2) Recursively split parent AABB along x-, y-, or z-axis (alternatively) by a split coordinate

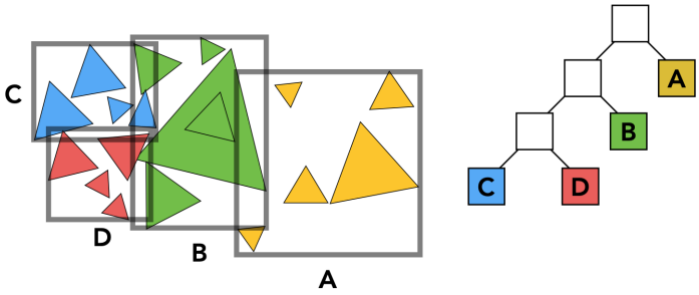
(3) Stop when the node is simple enough (leaf)
(4) Objects only stored in leaf nodes
(5) Ray query: from root; if ray hits AABB of the node, return the closer hit (from two children or stored objects); if not, ignore the node and its children



Problems:
- One object can be within multiple leaves
- Test intersection between objects and node AABBs is not easy

Bounding Volume Hierarchy (BVH, Object Partition)

(1) Find bounding box
(2) Recursively split set of objects in two subsets along one axis
(3) Recompute the bounding box of the subsets
(4) Stop when the node is simple enough (leaf)
(5) Objects only stored in leaf nodes
(6) Ray query: same as KD-Tree



Heuristic:
- Always choose the longest axis in node
- Split node at location of median object (O(n) find medium)
- Stop when node contains few elements (e.g., 5)

[Basic] Radiometry

Measurement system and units for illumination; perform lighting calculations in a physically correct manner

Name	Description	Symbol	Unit
Radiant Energy	Energy of electromagnetic radiation	Q	[J = Joule]
Radiant Flux (Power)	Energy emitted or received per unit time #photons flowing through a sensor in unit time	$\Phi \equiv \frac{\mathrm{d}Q}{\mathrm{d}t}$	[W = Watt] [lm = lumen]
Angle	Ratio of subtended arc length on circle to radius Circle has 2π radians	$\theta = \frac{l}{r}$	[radian]
Solid Angle	Radio of subtended area on sphere to radius squared Sphere has 4π steradians	$\omega = \frac{A}{r^2}$	[steradian]
Radiant Intensity	Power per unit solid angle	$I(\omega) \equiv \frac{\mathrm{d}\Phi}{\mathrm{d}\omega}$	$\left[\frac{\mathrm{W}}{\mathrm{sr}}\right]$

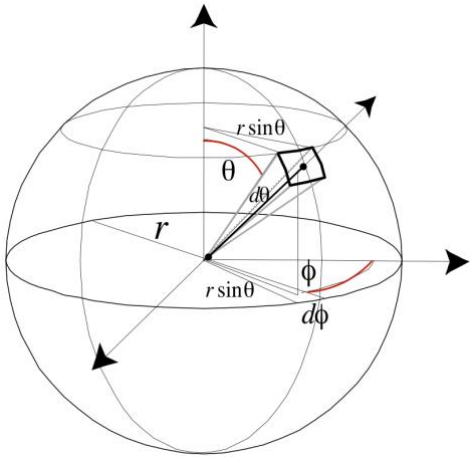
		$I = \frac{\Phi}{4\pi}$	$\left[\frac{\text{lm}}{\text{sr}} = \text{cd} = \text{candela}\right]$
Irradiance	Power per unit area Irradiance at a surface is proportional to cosine of angle between light direction and surface normal $\cos\theta = \mathbf{n} \cdot \mathbf{l}$	$E(p) = \frac{\mathrm{d}\Phi(p)}{\mathrm{d}A}$ $E = \frac{\Phi}{A}\cos\theta$	$\left[\frac{\text{W}}{\text{m}^2}\right]$ $\left[\frac{\text{lm}}{\text{m}^2} = \text{lux}\right]$
Radiance	Power per unit solid angle per projected unit area	$L(p,\omega)$ $= \frac{\mathrm{d}^2\Phi(p,\omega)}{\mathrm{d}\omega\ \mathrm{d}A\cos\theta}$	$\left[\frac{\text{W}}{\text{sr m}^2}\right]$ $\left[\frac{\text{cd}}{\text{m}^2} = \frac{\text{lm}}{\text{sr m}^2} = \text{nit}\right]$

Solid Angle Equations

$$\mathrm{d}A = (r\ \mathrm{d}\theta)(r\sin\theta\ \mathrm{d}\phi) = r^2\sin\theta\ \mathrm{d}\theta\ \mathrm{d}\phi$$

$$\mathrm{d}\omega = \frac{d\mathrm{A}}{r^2} = \sin\theta\ \mathrm{d}\theta\ \mathrm{d}\phi$$

$$\Omega = \int_{S^2} \mathrm{d}\omega = \int_0^{2\pi}\int_0^{\pi} \sin\theta\ \mathrm{d}\theta\ \mathrm{d}\phi = 4\pi$$



Irradiance vs. Radiance

Irradiance: total power received by area $\mathrm{d}A$

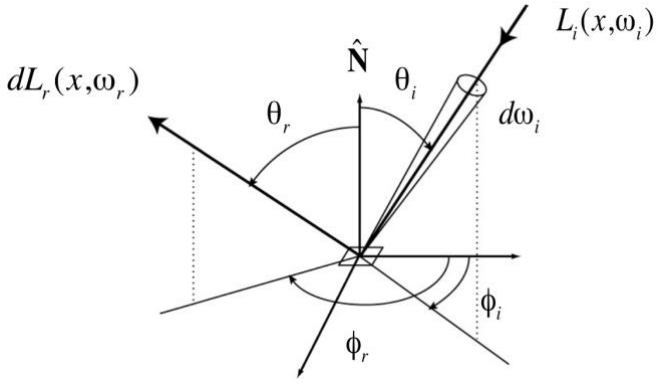
Radiance: total power received by area $\mathrm{d}A$ from "direction" $\mathrm{d}\omega$

$$\mathrm{d}E(p,\omega) = L_i(p,\omega)\cos\theta\ \mathrm{d}\omega$$

$$E(p) = \int_{H^2} L_i(p,\omega)\cos\theta\ \mathrm{d}\omega$$

Where H^2 is unit hemisphere

Bidirectional Reflectance Distribution Function (BRDF)



Radiance from direction ω_i turns into the power E that $\mathrm{d}A$ receives, then will become the radiance to any other direction ω_r

$$L_r(p,\omega_r) = \int_{H^2} f_r(p,\omega_i \to \omega_r) L_i(p,\omega_i) \cos\theta_i \,\mathrm{d}\omega_i$$

BRDF:

$$f_r(\omega_i \to \omega_r) = \frac{\mathrm{d}L_r(\omega_r)}{\mathrm{d}E_i(\omega_i)} = \frac{\mathrm{d}L_r(\omega_r)}{L_i(\omega_i)\cos\theta_i \,\mathrm{d}\omega_i} \quad \left[\frac{1}{\mathrm{sr}}\right]$$

Properties

- Non-negativity: $f_r(\omega_i \to \omega_r) \geq 0$
- Linearity: BRDF can be split and compute then sum together
- Reciprocity principle: $f_r(\omega_r \to \omega_i) = f_r(\omega_i \to \omega_r)$
- Energy conservation: $\int_{H^2} f_r(\omega_i \to \omega_r)\cos\theta_i \,\mathrm{d}\omega_i \leq 1$ for $\forall \omega_r$

BTDF: Bidirectional Transmition Distribution Function

BSDF: Bidirectional Scatter Distribution Function (BRDF + BTDF)

The Rendering Equation

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{\Omega^+} L_i(x,\omega_i) f_r(x,\omega_i,\omega_o)(\mathbf{n}\cdot\omega_i)\mathrm{d}\omega_i$$

Reflected light — Emission — Input light — BRDF — Input angle

Where $L_i(p,\omega_i)$ can be light source or reflected light from other objects: $L_o(x,-\omega_i)$

Fredholm Integral Equation of second kind with canonical form:

$$l(u) = e(u) + \int l(v)K(u,v)\mathrm{d}v$$

Discretized to a simple matrix equation

$$\begin{aligned} L &= E + KL && \text{vector = vector + matrix * vector} \\ &= (1-K)^{-1}E \\ &= (1 + K + K^2 + K^3 + \cdots)E && \text{Taylor expansion: } (1-x)^{-1} \end{aligned}$$

Global Illumination

$$L = \underbrace{E + KE}_{\text{Shading in rasterization}} + K^2E + K^3E + \cdots$$

- E: Emission directly from light source
- KE: Direct illumination on surface
- K^2E: Indirect illumination one bounce
- K^3E: Indirect illumination two bounces

Light bouncing will converge (e.g., K^8, K^{16})

[Basic] Monte Carlo Integration

Estimate the definite integral of given function

Sample on a **Probability Density Function (pdf)** $p(x)$ within $[a,b]$

$$\int_a^b f(x)\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)} \qquad X_i \sim p(x)$$

Path Tracing

$$L_o(x,\omega_o) = \int_{\Omega^+} L_i(x,\omega_i) f_r(x,\omega_i,\omega_o)\cos\theta_i \,\mathrm{d}\omega_i$$

$$\approx \frac{1}{N}\sum_{j=1}^{N}\frac{L_i(x,\omega_j)f_r(x,\omega_j,\omega_o)(\mathbf{n}\cdot\omega_j)}{p(\omega_j)}$$

(1) For one pixel, uniformly choose N sample position within the pixel, then average results
(2) For each sample, shoot a ray to hit the scene at x (one path)
(3) To shade $L(x,\omega_o)$, randomly choose one direction $\omega_i \sim p(\omega_i)$
(4) If the ray hits the light, return the radiance (direct, sampling the light); if the ray hits an object at y, recursively compute $L(y,-\omega_i)$ and return radiance (indirect)

Russian Roulette

Stop recurrence
Compute L_o without bias; not by limiting #bounce or energy
(1) Set a probability P $(0<P<1)$
(2) With probability P, shoot a ray and return the shading results by $\frac{L_o}{P}$
(3) With probability $1-P$, ignore and get 0

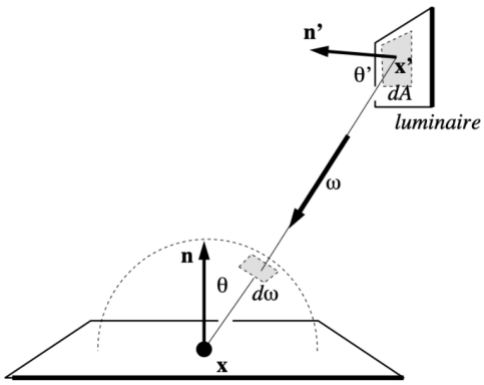
- $\mathrm{E}(L_o) = P*\frac{L_o}{P} + (1-P)*0 = L_o$
- Expected to stop at depth $\frac{P}{1-P}$ in recurrence

Sampling the Light

To avoid wasted rays when uniformly sampling the hemisphere at the shading point
Sampling the light doesn't need Russian Roulette
Need to test if the ray is blocked by other objects

$$\mathrm{d}\omega = \frac{\mathrm{d}A\cos\theta'}{\|x'-x\|^2}$$

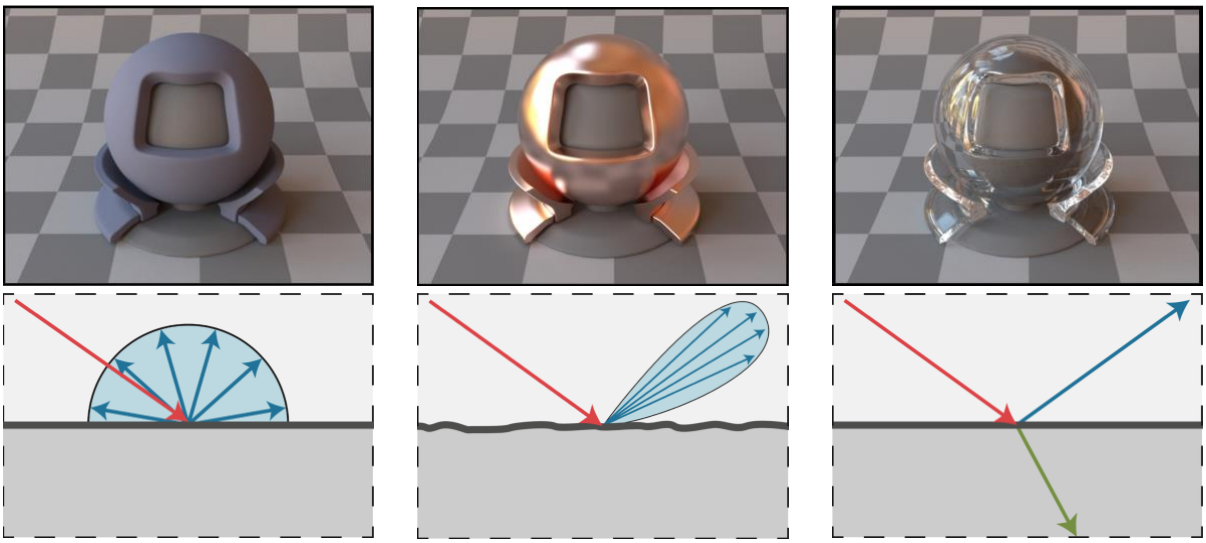
$$L_o(x,\omega_o) = \int_{\Omega^+} L_i(x,\omega_i)f_r(x,\omega_i,\omega_o)\cos\theta\,\mathrm{d}\omega_i$$
$$= \int_A L_i(x,\omega_i)f_r(x,\omega_i,\omega_o)\frac{\cos\theta\cos\theta'}{\|x'-x\|^2}\mathrm{d}A$$



Other Notes

- Sample at pixel instead of sample ω at point to avoid recursive exponential explosion
- Radiance to color needs gamma correction
- How to sample the hemisphere? What's the best choice of pdfs? (importance sampling)
- Combing sampling both the hemisphere and the light (multiple importance sampling)
- Why radiance of a pixel is the average on all paths passing on it

Material



Diffuse/Lambertian Material | Glossy Material | Ideal Reflective/Refractive Material

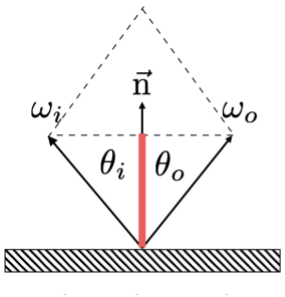
Diffuse (Lambertian) Material

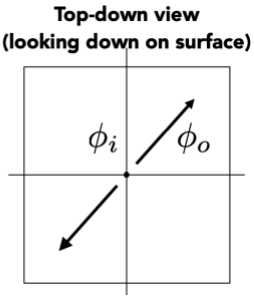
Light is equally reflected in each output direction

$$f_r = \frac{\rho}{\pi}$$

Where ρ is albedo (color)

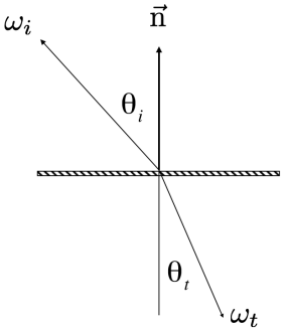
Perfect Specular Reflection

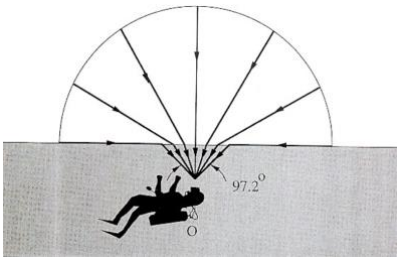




$$\theta = \theta_i = \theta_o$$
$$\phi_o = (\phi_i + \pi) \bmod 2\pi$$
$$\omega_o + \omega_i = 2\cos\theta\,\vec{\mathbf{n}} = 2(\omega_i \cdot \vec{\mathbf{n}})\vec{\mathbf{n}}$$
$$\omega_o = -\omega_i + 2(\omega_i \cdot \vec{\mathbf{n}})\vec{\mathbf{n}}$$

[Basic] Snell's Law





$$\eta_i \sin\theta_i = \eta_t \sin\theta_t$$

$$\cos\theta_t = \sqrt{1 - \sin^2\theta_t}$$

$$= \sqrt{1 - \left(\frac{\eta_i}{\eta_t}\right)^2 \sin^2\theta_i}$$

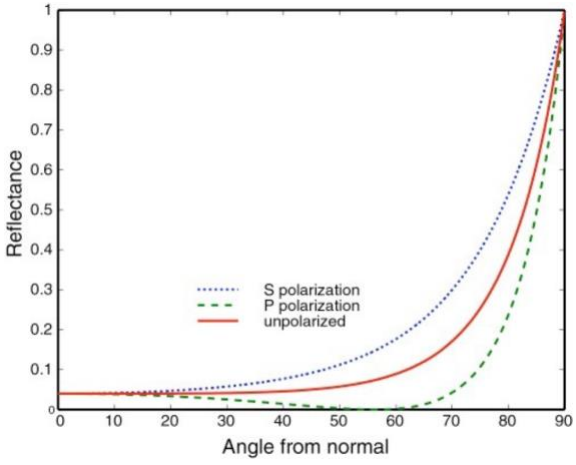
When $\frac{\eta_i}{\eta_t} > 1$, $\cos\theta_t$ may not exist: light incident on boundary from large enough angle (**grazing angle**) will not refract

Index of refraction is wavelength dependent

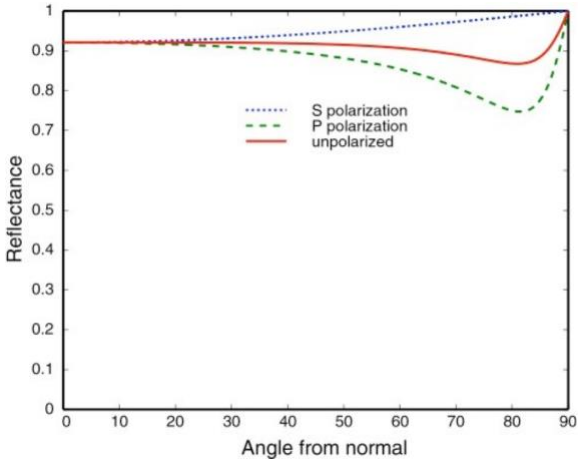
Medium	$\boldsymbol{\eta}$
Vacuum	1.0
Air (sea level)	1.00029
Water (20℃)	1.333
Glass	1.5-1.6
Diamond	2.42

[Basic] Fresnel Term / Reflection

Reflectance depends on incident angle (and polarization of light)



Dielectric, $\eta = 1.5$



Conductor

Consider polarization:

$$R_s = \left|\frac{\eta_1 \cos\theta_i - \eta_2 \cos\theta_t}{\eta_1 \cos\theta_i + \eta_2 \cos\theta_t}\right|^2$$

$$R_t = \left|\frac{\eta_1 \cos\theta_t - \eta_2 \cos\theta_i}{\eta_1 \cos\theta_t + \eta_2 \cos\theta_i}\right|^2$$

$$R_{\text{eff}} = \frac{1}{2}(R_s + R_t)$$

Schlick's approximation:

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5$$

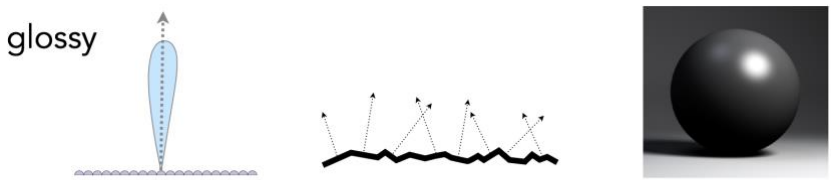
$$R_0 = \left(\frac{\eta_1 - \eta_2}{\eta_1 + \eta_2}\right)^2$$

Microfacet Material

Surface from different views:

- From macroscale view: flat & rough => Texture
- From microscale view: bumpy & specular => Geometry

Microfacet theory: individual element of surface act like mirrors (microfacets), each microfacet has its own normal



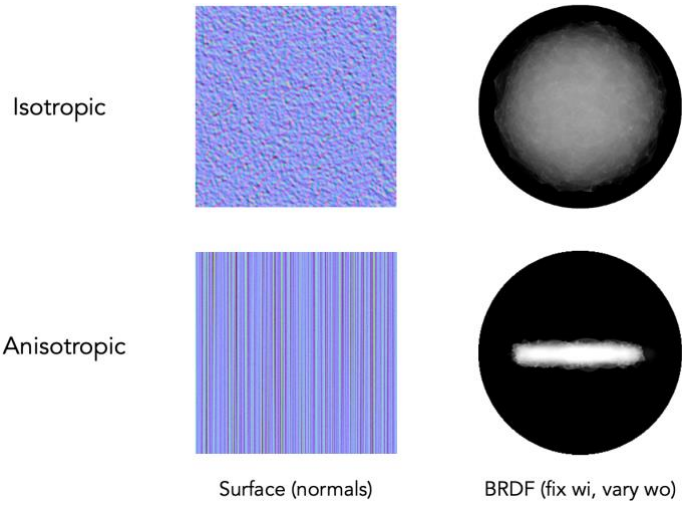
BRDF: the distribution of microfacets' normal

$$f(\mathbf{i},\mathbf{o}) = \frac{\overbrace{\mathbf{F}(\mathbf{i},\mathbf{h})}^{\text{Fresnel term}}\ \overbrace{\mathbf{G}(\mathbf{i},\mathbf{o},\mathbf{h})}^{\text{shadowing-masking term}}\ \overbrace{\mathbf{D}(\mathbf{h})}^{\text{distribution of normals}}}{4(\mathbf{n}\cdot\mathbf{i})(\mathbf{n}\cdot\mathbf{o})}$$

Where $\mathbf{h} = \frac{\mathbf{1}}{\mathbf{2}}(\mathbf{i}+\mathbf{o})$ is half vector

Shadowing-masking term: consider shadowing between micro bumps

Isotropic / Anisotropic Materials



Isotropic BRDF (reduce dimensionality from 4D to 3D):

$$f_r(\theta_i,\phi_i;\theta_r,\phi_r) = f_r(\theta_i,\theta_r,\phi_r-\phi_i) = f_r(\theta_i,\theta_r,|\phi_r-\phi_i|)$$

Anisotropic BRDFs:

$$f_r(\theta_i,\phi_i;\theta_r,\phi_r) \neq f_r(\theta_i,\theta_r,\phi_r-\phi_i)$$

Computational Photography

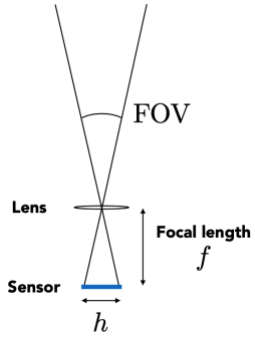
Pinhole Image Formation

Light passes through a small hole to generate an inversed image

Why pinhole?: larger hole will make each pixel on image (sensor) integrate all lights from the object, so all pixel values would be similar

Each pixel records irradiance

Field of View (FOV)



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

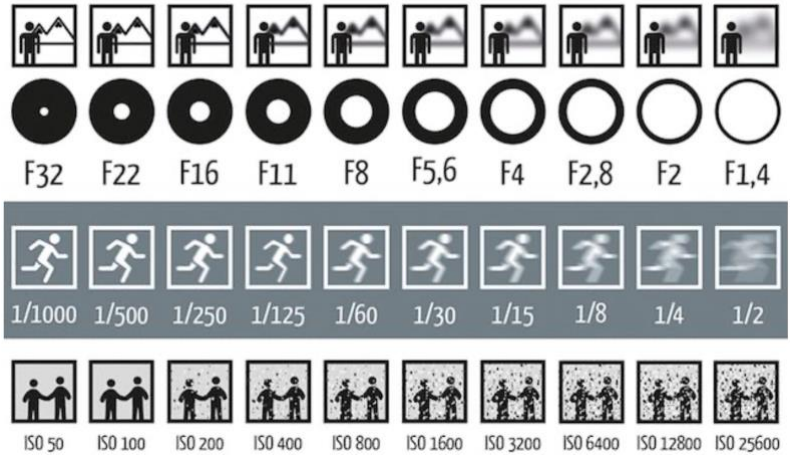
To maintain FOV, decrease focal length of lens in proportion to width/height of sensor
For historical reasons, it is common to refer to angular field of view by focal length of a lens used on a 35mm-format film (36 x 24mm):

- 17mm is wide angle 104°
- 50mm is a "normal" lens 47°
- 200mm is telephoto lens 12°

Exposure

Exposure = time × irradiance (H=T×E)
T (exposure time): controlled by shutter
E (irradiance): power of light falling on a unit area of sensor, controlled by lens aperture and focal length
Exposure control in photography:

- Aperture size: change the f-stop by opening / closing the aperture (if camera has iris control);
- Shutter speed: change the duration the sensor pixels integrate light
- ISO gain: change the amplification (analog and/or digital) between sensor values and digital image values



F-Number (F-Stop)

Exposure levels
Definition: the focal length divided by the diameter of the aperture
Written as FN, FN,M (=FN.M) or F/N; N is the f-number
Smaller N, larger aperture size, more blur

Shutter Speed

Slower shutter speed, more motion blur: e.g., hand shake, subject movement

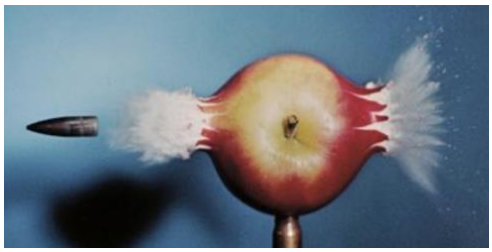
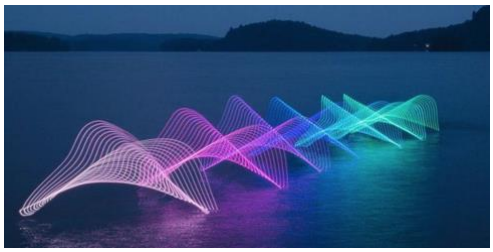
Double shutter time doubles motion blur
Rolling shutter problem: different parts of photo taken at different times, cause deformation

ISO Gain

Film: trade sensitivity for grain
Digital: trade sensitivity for noise
(1) Multiply signal before analog-to-digit conversion
(2) Linear effect (ISO 200 needs half the light as ISO 100)
More ISO, more noise

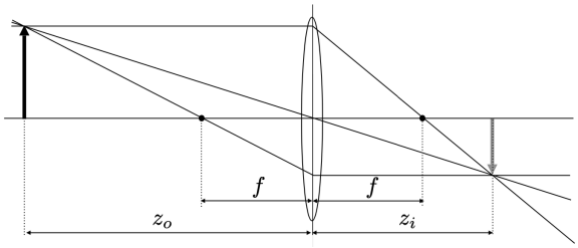
Fast and Slow Photography

High-Speed Photography: extremely fast shutter speed, larger aperture and/or high ISO
Long-Exposure Photography: extremely slow shutter speed

Thin Lens

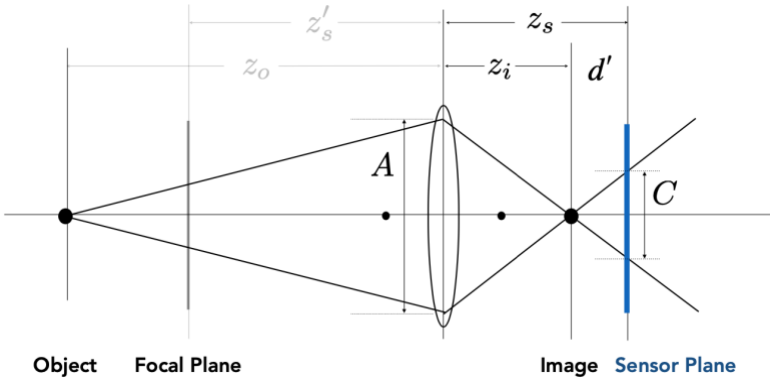
- All parallel rays entering a lens pass through its focal point
- All rays through a focal point will be in parallel after passing the lens
- Focal length f can be arbitrarily changed



Thin lens equation:

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

Circle of Confusion (CoC)



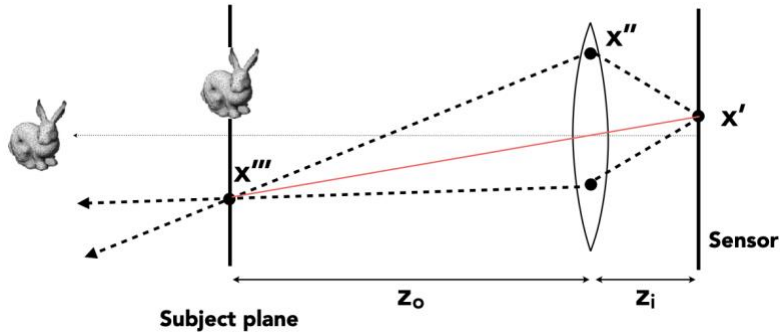
Circle of confusion is proportional to the size of the aperture

$$\frac{C}{A} = \frac{d'}{z_i} = \frac{|z_s - z_i|}{z_i}$$

Ray Tracing Ideal Thin Lenses

One possible setup:

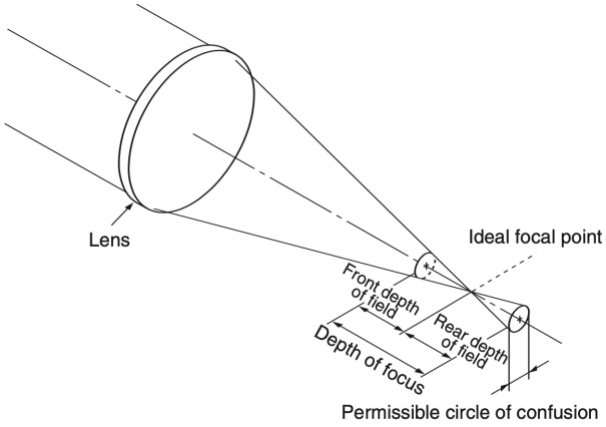
(1) Choose sensor size, lens focal length and aperture size
(2) Choose depth of subject of interest z_o
(3) Calculate corresponding depth of sensor z_i from thin lens equation
(4) For each pixel x' on the sensor, sample random point x'' on lens plane
(5) Pass the ray through the lens will hit x''' (consider virtual ray x' -> x''')
(6) Estimate radiance on ray x'' -> x'''

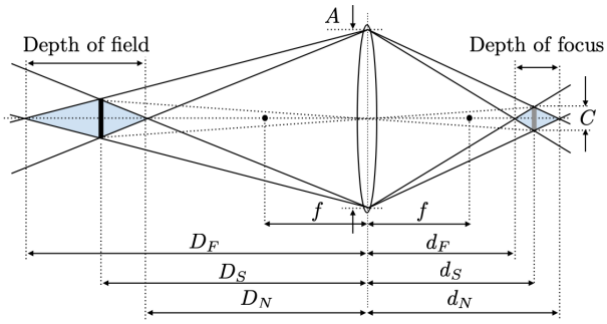


Depth of Field

Set circle of confusion as the maximum permissible blur spot on the image plane that will appear sharp under final viewing conditions

$$\mathrm{DOF} = \mathrm{DF} - \mathrm{DN} = \frac{D_S f^2}{f^2 - \frac{f}{A} C(D_S - f)} - \frac{D_S f^2}{f^2 + \frac{f}{A} C(D_S - f)}$$





The Plenoptic Function

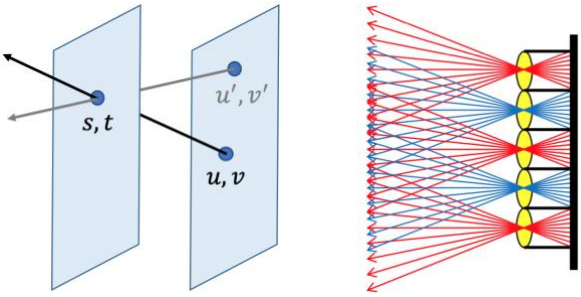
$$P(\theta, \phi, \lambda, t, V_x, V_y, V_z)$$

Can reconstruct the intensity of light from every possible view (θ, ϕ), at every moment (t), from every position (V_x, V_y, V_z), at every wavelength (λ)

Contains every photograph, every movie, everything that anyone has ever seen

Light Field (Lumigraph)

Store the radiance information of every pixel seen from every direction (2D+2D=4D)
2-plane parameterization (2D position + 2D position): (s, t, u, v)



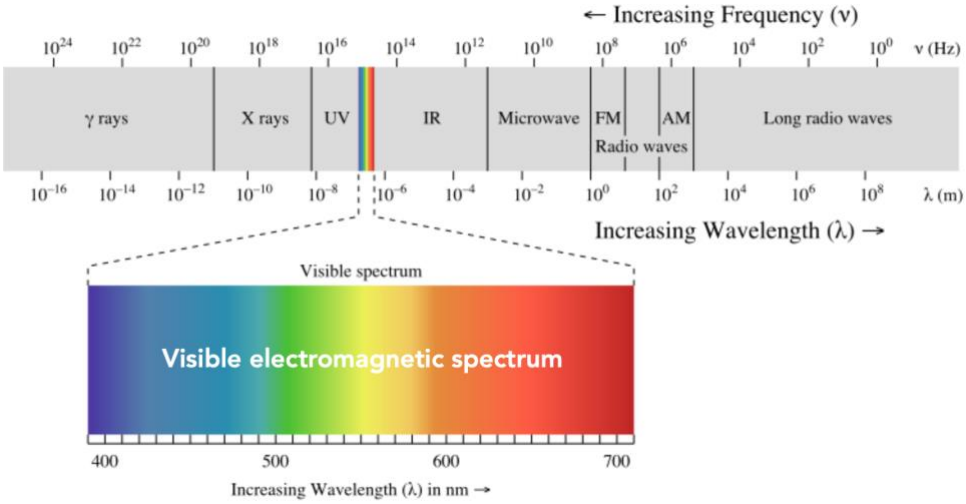
Light field camera: each pixel (irradiance) is stored as a block of pixels (radiance)

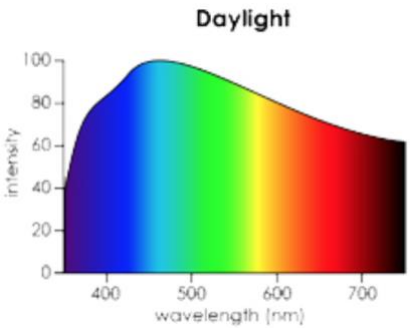
- Two-layer structure: microlens on each pixel (1st layer) transmit lights from different directions to different 2D locations on digital sensor (2nd layer)
- Virtually change focal length & aperture size, etc. after taking photo
- Insufficient spatial resolution

Color

Color is a phenomenon of human perception; it is not a universal property of light

Spectral Power Distribution (SPD)





Salient property in measuring light; the amount of light present at each wavelength
Linear additive

The Human Eye

Retinal photoreceptor cell:

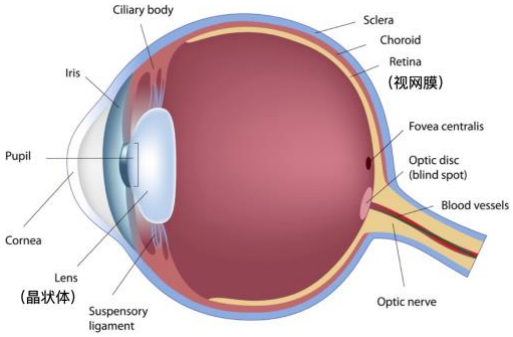
- Rod: primary receptors in very low light ("scotopic" conditions, e.g. dim moonlight); ~120 million rods in eye; perceive only shades of gray, no color
- Cone: primary receptors in typical light levels ("photopic"); ~6-7 million cones in eye; provide sensation of color; three types of cones (S,M,L), each with different spectral sensitivity

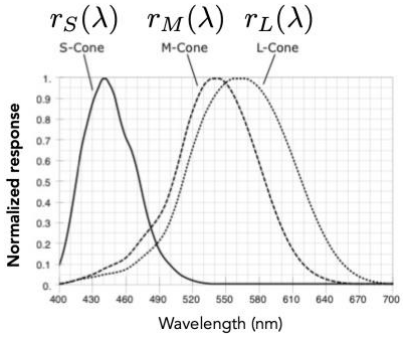
Assume $\mathbb{S}(\lambda)$ is spectrum (SPD), then the responses:

$$S = \int_{\lambda} r_S(\lambda) \cdot \mathbb{S}(\lambda)\, \mathrm{d}\lambda$$

$$M = \int_{\lambda} r_M(\lambda) \cdot \mathbb{S}(\lambda)\, \mathrm{d}\lambda$$

$$L = \int_{\lambda} r_L(\lambda) \cdot \mathbb{S}(\lambda)\, \mathrm{d}\lambda$$





The eye "sees" only three response values (S,M,L), and this is only info available to brain

Metamerism: two different spectra that project to the same (S,M,L) response

CIE XYZ

Imaginary set of standard color primaries X,Y,Z

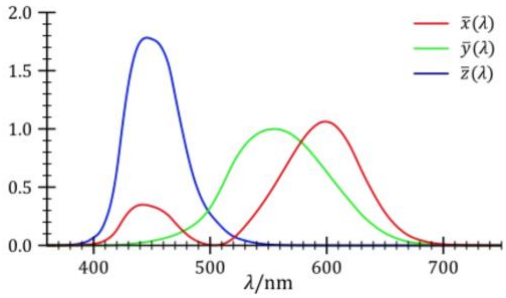
- Primary colors with these matching functions do not exist
- Y is luminance (brightness regardless of color)
- Matching functions are strictly positive; span all observable colors

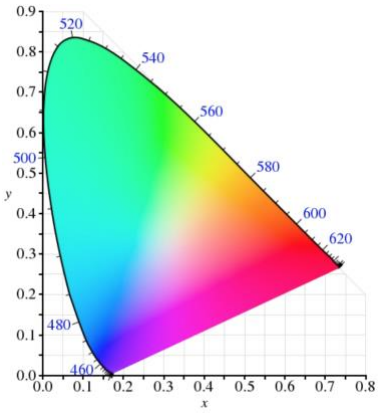
Color Matching Function

$$X_{\text{CIE XYZ}} = \int_{\lambda} \bar{x}(\lambda) \cdot \mathbb{S}(\lambda)\, \mathrm{d}\lambda$$

$$Y_{\text{CIE RGB}} = \int_{\lambda} \bar{y}(\lambda) \cdot \mathbb{S}(\lambda)\, \mathrm{d}\lambda$$

$$Z_{\text{CIE RGB}} = \int_{\lambda} \bar{z}(\lambda) \cdot \mathbb{S}(\lambda)\, \mathrm{d}\lambda$$





Chromaticity Diagram

(1) Choose a fixed luminance Y

(2) Chromaticity defined as

$$x = \frac{X}{X+Y+Z},\quad y = \frac{Y}{X+Y+Z},\quad z = \frac{Z}{X+Y+Z}$$

(3) Since $x + y + z = 1$, plot (x, y) cords at specific brightness Y

- Curve boundary corresponds to monochromatic light (pure color of a single wavelength)
- Any color inside is less pure (mixed)

Gamut

Gamut is the set of chromaticities generated by a set of color primaries

Different color spaces represent different ranges of colors

CIE RGB

Additive Color Space: more color mixed, lighter it will be; describe light

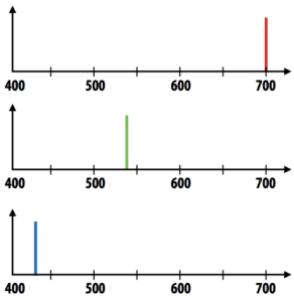
(1) Given a set of primary lights, each with its own spectral distribution (e.g. R,G,B-single wavelength)

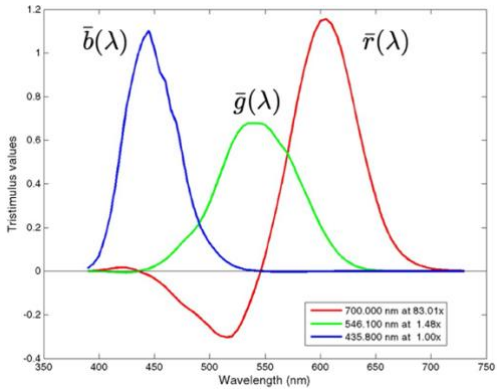
(2) The brightness of lights can be added together

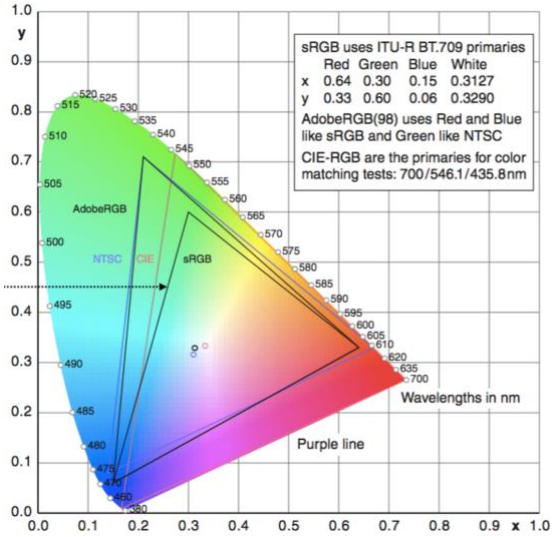
$$R \cdot \mathbb{S}_R(\lambda) + G \cdot \mathbb{S}_G(\lambda) + B \cdot \mathbb{S}_B(\lambda)$$

(3) Color (with wavelength λ) can be measured and described by the scalar values

$$R, G, B = \bar{r}(\lambda), \bar{g}(\lambda), \bar{b}(\lambda)$$







sRGB

A particular monitor RGB standard

Gamut is limited

Gamma Correction: nonlinear transfer between the intensity and actual values stored, to facilitate the luminance discrimination of human (a little darker than linear RGB)

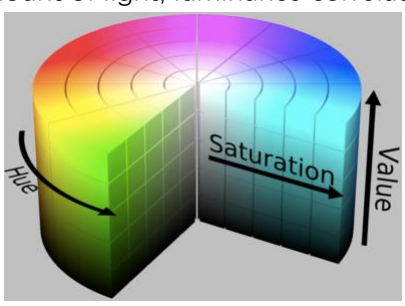
$$\text{linearRGB} \rightarrow \text{sRGB}: \quad \gamma(u) = \begin{cases} 12.92u & u \leq 0.0031308 \\ 1.055u^{\frac{1}{2.4}} - 0.055 & \text{otherwise} \end{cases}$$

$$\text{sRGB} \rightarrow \text{linearRGB}: \quad \gamma^{-1}(v) = \begin{cases} \dfrac{v}{12.92} & v \leq 0.04045 \\ \left(\dfrac{v + 0.055}{1.055}\right)^{2.4} & \text{otherwise} \end{cases}$$

HSV

Perceptually organized color space

- Hue: the "kind" of color; dominant wavelength correlated
- Saturation: the "colorfulness"; purity correlated
- Lightness (or Value): the overall amount of light; luminance correlated



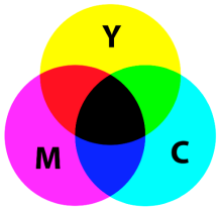
CIE Lab

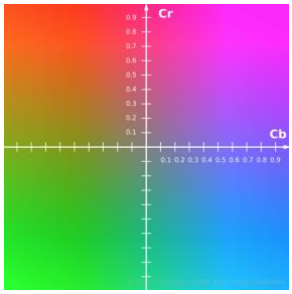
Opponent Color Theory: the brain seems to encode color early on using three axes: white-**black**, **red**-**green**, **blue**-**yellow** (e.g., evidence: visual residual)

- L: lightness
- a & b: color-opponent pair; a (**red**-**green**), b (**blue**-**yellow**)

CMYK

Subtractive Color Space: more color mixed, darker it will be; describe material which absorb light
Cyan, Magenta, Yellow and Key (Black, for black-and-white-print to save chromatic ink)





YUV (YCbCr)

- Y: luminance
- U & V: chorma

Human is sensitive to luminance rather than location and chroma, so use chrominance subsampling to resample U & V for depression (often 2, 4, or 8 pixels shared the same U + V)

High-Dynamic Range (HDR)

A greater range of luminosity than standard digital imaging (16-bit data)
Useful for recording very bright or dark scenes, providing detailed resolution around bright/dark color
Result from merging multiple low-dynamic range (LDR) photographs

Animation

Mass Spring System

Force pulls points at length l:

$$\boldsymbol{f}_{\mathrm{a}} = k_s \quad \frac{\boldsymbol{b}-\boldsymbol{a}}{\|\boldsymbol{b}-\boldsymbol{a}\|} \quad (\|\boldsymbol{b}-\boldsymbol{a}\|-l) = -\boldsymbol{f}_{\mathrm{b}}$$

Stiffness Direction Rest length

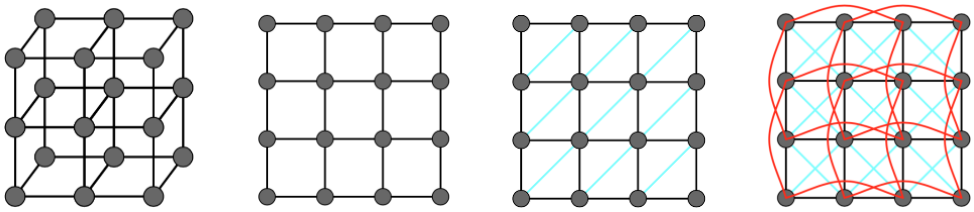
Energy loss:

$$\boldsymbol{f}_{\mathrm{a}} = k_d \quad \frac{\boldsymbol{b}-\boldsymbol{a}}{\|\boldsymbol{b}-\boldsymbol{a}\|} \cdot (\dot{\boldsymbol{b}}-\dot{\boldsymbol{a}}) \quad \cdot \quad \frac{\boldsymbol{b}-\boldsymbol{a}}{\|\boldsymbol{b}-\boldsymbol{a}\|}$$

Damping | Relative velocity projected | Direction

Where $\boldsymbol{a}$ is position, $\dot{\boldsymbol{a}} = \frac{\mathrm{d}\boldsymbol{a}}{\mathrm{d}t}$, $\ddot{\boldsymbol{a}} = \frac{\mathrm{d}\dot{\boldsymbol{a}}}{\mathrm{d}t}$

Structures from springs:



Blocks | Sheets | Resist shearing | Resist bending

Particle System

Model dynamical systems (e.g., fire, fluid, steam) as collections of large numbers of particles

1. For each frame, create new particles
2. Calculate forces on each particle, update its position and velocity
3. Remove dead particles, render

Particle forces:

- Attraction and repulsion forces: gravity, electromagnetism, springs, propulsion …

$$\boldsymbol{f}_{\mathrm{g}} = G\frac{m_1 m_2}{d^2}$$

- Damping forces: friction, air drag, viscosity
- Collisions

Simulated as an **Ordinary Differential Equation (ODE)**

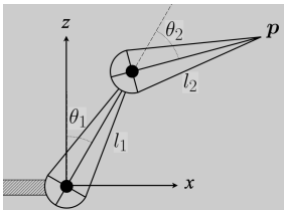
Forward Kinematics

Articulated skeleton

- Topology & tree structure
- Geometric relations from joints: pin (1D rotation), ball (2D rotation), prismatic (translation)

Convenient for direct control, not convenient for artists

$$p_z = l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2)$$
$$p_x = l_1\sin(\theta_1) + l_2\sin(\theta_1+\theta_2)$$



Inverse Kinematics

Animator provides position of end-effector, computer determine joint angles that satisfy constraints

$$\theta_2 = \cos^{-1}\left(\frac{p_z^2 + p_x^2 - l_1^2 - l_2^2}{2l_1 l_2}\right)$$

$$\theta_1 = \frac{-p_z l_2\sin(\theta_2) + p_x(l_1 + l_2\cos(\theta_2))}{p_x l_2\sin(\theta_2) + p_z(l_1 + l_2\cos(\theta_2))}$$

May have multiple solutions or no solution
Numerical solution to general N-link IK problem:
1. Choose an initial configuration
2. Define an error metric (e.g., square of distance between goals and current position)
3. Compute gradient of error as function of configuration
4. Apply gradient descent (or Newton's method, or other optimization procedures)

Key-Based Animation Generation

Keyframe Animation

Lead animator creates keyframes; computer creates in-between frames (tweening)
Consider each keyframe as a vector with parameters, then interpolate (e.g., splines)

Rigging

High-level control on characters to allow rapid and intuitive modification of pose, deformations, expression, etc.
Manual control key points / curves / etc.

Blending Shapes

Instead of skeleton, interpolate directly between surfaces (e.g., model a collection of facial expressions)

Motion Capture

Record real-world performance (e.g., person executing and activity)
High realism, complex set-ups, require artists' alternations
Equipment:
- Optical: 3D positional tracking by triangulation from multiple cameras; occlusions are difficult
- Magnetic (sense magnetic field to infer position/orientation)
- Mechanical (measure joint angle directly, restrict motion)

Euler's Method

Single particle simulation by measuring position and velocity
Default is Explicit Euler (Forward Euler)

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

With numerical integration, errors accumulate
- Inaccuracy increases as Δt increases
- Instability, causes simulation to diverge

Midpoint Method

$$\boldsymbol{x}_{\text{mid}} = \boldsymbol{x}(t) + \frac{\Delta t}{2}\boldsymbol{v}(\boldsymbol{x}(t), t)$$
$$\boldsymbol{x}(t+\Delta t) = \boldsymbol{x}(t) + \Delta t\, \boldsymbol{v}(\boldsymbol{x}_{\text{mid}}, t)$$

Modified Euler

Average velocity at start and end of step

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

Adaptive Step Size

1. Compute $\boldsymbol{x}_1^T$ with a Euler step of size T; compute $\boldsymbol{x}_{1/2}^T$ with two Euler steps of size $T/2$
2. Compute error $\left\|\boldsymbol{x}_1^T - \boldsymbol{x}_{1/2}^T\right\|$, if error exceeds a threshold reduce step size and try again

Implicit Euler (Backward Euler)

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

Solve nonlinear problem by root-finding algorithm (e.g., Newton's method)
Much better stability:

- Implicit Euler has order-one
- Local truncation error: $O((\Delta t)^2)$
- Global truncation error: $O(\Delta t)$

Runge-Kutta Families

Solving ODEs, good at dealing with non-linearity
Its order-four version is the most widely used (a.k.a. RK4):

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y),\ \ y(t_0) = y_0$$
$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{n+1} = t_n + h$$
$$k_1 = f(t_n,\ \ y_n)$$
$$k_2 = f\left(t_n + \frac{h}{2},\ \ y_n + h\frac{k_1}{2}\right)$$
$$k_3 = f\left(t_n + \frac{h}{2},\ \ y_n + h\frac{k_2}{2}\right)$$
$$k_4 = f(t_n + h,\ \ y_n + hk_3)$$

Verlet Integration

Four-order precision to simulate position

$$x(t + \Delta t) = x(t) + \dot{x}(t)\Delta t + \frac{1}{2}\ddot{x}(t)(\Delta t)^2 + \frac{1}{6}\dddot{x}(t)(\Delta t)^3 + O((\Delta t)^4)$$
$$x(t - \Delta t) = x(t) - \dot{x}(t)\Delta t + \frac{1}{2}\ddot{x}(t)(\Delta t)^2 - \frac{1}{6}\dddot{x}(t)(\Delta t)^3 + O((\Delta t)^4)$$

Taylor expansion and sum

$$x(t + \Delta t) = 2x(t) - x(t - \Delta t) + \ddot{x}(t)(\Delta t)^2 + O((\Delta t)^4)$$

Velocity of t can be get at time $t + \Delta t$

$$v(t) = \frac{x(t + \Delta t) - x(t - \Delta t)}{2\Delta t} + O((\Delta t)^2)$$

Rigid Body Simulation

No internal deformation; similar to simulate a particle

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} \mathbf{X} & \dot{\mathbf{X}} & \theta & \omega \end{pmatrix} = \begin{pmatrix} \dot{\mathbf{X}} & \frac{\mathbf{F}}{m} & \omega & \frac{\Gamma}{I} \end{pmatrix}$$

Position Velocity Angle Angular velocity Force / mass Torque / momentum of inertia

Fluid Simulation

Simple **Position-Based Method** / **Verlet Integration**

1. Assume water is composed of small rigid-body spheres and cannot be compressed
2. As long as density changes somewhere, it should be corrected via changing the positions of particles
3. Compute gradient of the density anywhere w.r.t. each particle's position
   Density: near neighbor problem
4. Gradient descent

Lagrangian vs. Eulerian

Two different views to simulate large collections of objects
Lagrangian (mass point) approach: monitor the same particle through out its course
Eulerian (grid) approach: monitor a stationary region and consider particles pass through

Material Point Method (MPM)

Hybrid, combining Eulerian and Lagrangian views

- Lagrangian: consider particles carrying material properties
- Eulerian: use a grid to do numerical integration
- Interaction: particles transfer properties to grid, grid performs update, then interpolate back to particles

[Advanced] Rendering

Bidirectional Path Tracing (BDPT)

- Traces sub-paths from both the camera and the light
- Connects the end points from both sub-paths

- Unbiased
- Suitable if the light transport is complex on the light's side
- Difficult to implement & quite slow

Metropolis Light Transport (MLT)

Markov Chain Monte Carlo (MCMC): Jumping from the current sample to the next with specific PDF
Achieve minimum variance when $f(x)$ and $p(x)$ are of the same shape in Monte Carlo Integration

- Locally perturb an existing path to get a new path

- Unbiased
- Very good at locally exploring difficult light paths
- Works great with difficult light path
- Difficult to estimate the convergence rate, not used to render animation

- Does not guarantee equal convergence rate per pixel (dirty results)

Photon Mapping

1. Stage 1 (photon tracing): emitting photons from the light source, bouncing them around, then recording photons on diffuse surfaces
2. Stage 2 (photon collection, final gathering): shoot sub-paths from the camera, bouncing them around, until they hit diffuse surfaces
3. Calculation (local density estimation): for each shading point, find the nearest N photons; take the surface area A they cover; brightness is N/A

- More photons emitted => the same N photons cover smaller ΔA => ΔA is closer to dA
- Biased (blurry), but consistent (not blurry with infinite #samples)
- Very good at handling **Specular-Diffuse-Specular (SDS)** paths and generating **caustics**

Vertex Connection and Merging (VCM)

Combination of BDPT and Photon Mapping

- Not waste the sub-paths in BDPT if their end points cannot be connected but can be merged (close)
- Use photon mapping to handle the merging of nearby "photons"

Instant Radiosity (IR)

Many-light approaches: lit surfaces can be treated as light sources

1. Shoot light sub-paths and assume the end point of each sub-path is a Virtual Point Light (VPL)
2. Render the scene as usual using these VPLs

- Fast and usually gives good results on diffuse scenes
- Spikes will emerge when VPLs are close to shading points
- Cannot handle glossy materials

[Advanced] Material

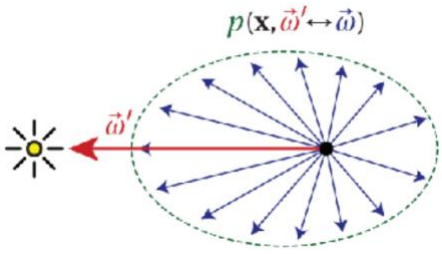
Participating Media

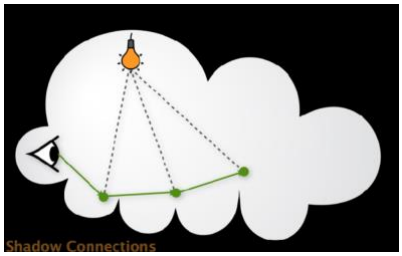
Like cloud, fog, ···

At any point as light travels through a participating medium, it can be (partially) absorbed and scattered

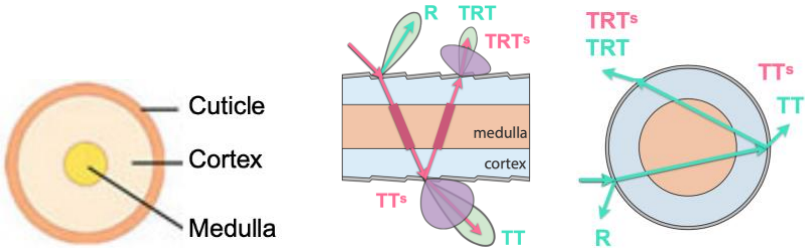
Use Phase Function to describe the angular distribution of light scattering at any point x within participating media

- Randomly choose a direction to bounce
- Randomly choose a distance to go straight
- At each "shading point", connect to the light





Hair Appearance



Kajiya-Kay Model

Scatter the incident light into a cone-shaped region

Marschner Model

Treat hair as glass-like cylinder
3 types of light interactions: R, TT and TRT (R: reflection; T: transmission)

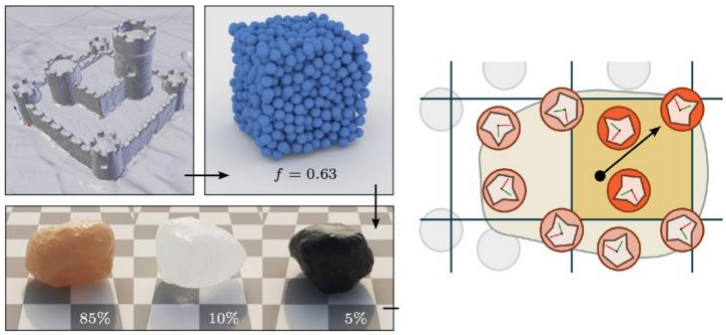
Yan's Model

Treat hair as double cylinder model
5 types of light interactions: plus TRT^s and TT^s

- Good to represent diffusive and saturated appearance (e.g., animal fur)

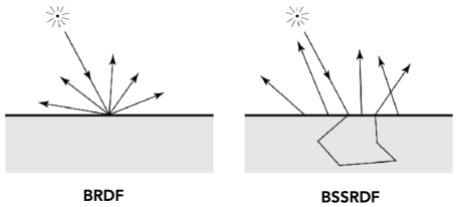
Granular Material

Procedural definition to avoid explicit modeling of all granules



Subsurface Scattering

Rendering translucent surface
Visual characteristics of many surfaces caused by light exiting at different points than it enters

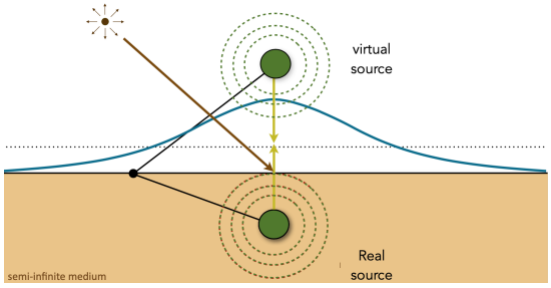


BSSRDF: $S(x_i, \omega_i, x_o, \omega_o)$; generalization of BRDF; exit radiance at one point due to incident differential irradiance at another point
Scattering function:

$$L_o(x, \omega_o) = \int_A \int_{H^2} S(x_i, \omega_i, x_o, \omega_o) L_i(x, \omega_i) \cos\theta_i \, \mathrm{d}\omega_i \mathrm{d}A$$

Dipole Approximation

Approximate light diffusion by introducing two point sources



Cloth

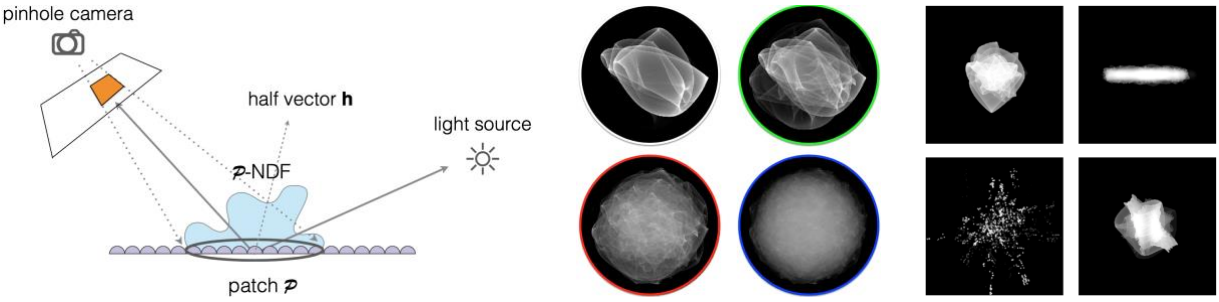
Fiber: a slender, elongated, threadlike unit
Ply: collection of twisted fibers
Yarn: collection of twisted plies
Cloth: woven or knitted yarns

- Given the weaving pattern; render as surface; using BRDF
- Properties of individual fibers & their distribution; render as a participating medium (velvet)
- Render every fiber explicitly

Detailed Appearance

Real world is more complicated; ideal rendering is not realistic

- Use actual distribution of normals (NDF) instead of statistical distribution (with no noise)
- BRDF over a pixel (corresponding to a larger patch) to generate noisy NDF



- Recent trend: detailed material under wave optics

Procedural Appearance

<3D Procedural Noise + Solid Modeling>
Complex noise function can generate 3D texture: maintains, water, ···