CS184 Notes

Triangles

For a triangle defined by points $\{P_1, P_2, P_3\}$ (going clockwise), let $P_i = (X_i, Y_i)$, $dX_i = X_{i+1} - X_i$, $dY_i = Y_{i+1} - Y_i$. The line equations of the three edges are

$$L_i(x, y) = -(x - X_i)dY_i + (y - Y_i)dX_i$$

A sample point (x', y') is inside the triangle if $L_1(x', y'), L_2(x', y'), L_3(x', y') > 0$. *Note that ≥ 0 must be used to handle edge cases.*

Aliasing refers to artifacts that result from undersampling (two frequencies that are indistinguishable at a given sampling rate are called aliases).

Anti-aliasing is the process of removing frequencies above the Nyquist frequency (twice the highest frequency present) before sampling.

Supersampling is an anti-aliasing technique where $N \times N$ samples are taken inside the pixel (not just at the center) to get an average color value, resulting in smoother edge transitions.

Convolution theorem: Convolution in the spatial domain is equal to multiplication in the frequency domain.

We can visualize frequency space with a 2D log magnitude image. High-frequency patterns make bright areas away from the origin; low-frequency features light up the origin. Horizontal frequency oscillations appear on the x-axis and vertical frequency oscillations appear on the y-axis.

Transforms

Transformation Order

- Object coordinates
- World (scene) coordinates: Apply viewing transform
- Camera (eye) coordinates: Apply perspective transform and homogeneous division
- Normalized device coordinates: Apply 2D screen transform
- Screen coordinates: Rasterization

Homogeneous coordinates: 3D points are represented as (x, y, z, w) where $w \in \{0, 1\}$.

Affine transforms are linear transforms plus translations. Example 2D transforms:

Scaling

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Translation

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation (cc)

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Coordinate Transform

$$\begin{pmatrix} \mathbf{u} & \mathbf{v} & \mathbf{o} \\ 0 & 0 & 1 \end{pmatrix}$$

where $\mathbf{u}, \mathbf{v}, \mathbf{o} \in \mathbb{R}^2$ are the new $\mathbf{e_1}$, $\mathbf{e_2}$ (relative to the new origin), and origin vectors, resp.

3D Rotation by angle α about axis $\mathbf{n}$:

$$\cos(\alpha)I + (1 - \cos(\alpha))\mathbf{n}\mathbf{n}^\top + \sin(\alpha)\begin{pmatrix} 0 & n_z & -n_y \\ -n_z & 0 & n_x \\ n_y & -n_x & 0 \end{pmatrix}$$

Standard camera: Located at origin, looking down negative z-axis, up vector is y-axis. Given world space eye point $\mathbf{e}$, view direction $\mathbf{v}$, up vector $\mathbf{u}$, the world-camera viewing transform is

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

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

where $\mathbf{r} = \mathbf{v} \times \mathbf{u}$.

Projective transform: Project scene point (x, y, z) onto an image plane, with the center of projection at the origin and image plane at $z = d$, with $(x, y, z) \rightarrow (xd/z, yd/z, d)$.

Perspective projection: Perspective viewing volume parameterized by *fovy*: vertical angular field of view; *aspect*: width/height of field of view; *near*: depth of near clipping plane; and *far*: depth of far clipping plane.

Derived quantities: $top = near \cdot \tan(fovy)$, $bottom = -top$, $right = top \cdot aspect$, $left = -right$.

Convert from camera coordinates to normalized device coordinates (NDC) by mapping the view volume frustum into a cube, where $(left, bottom, -near) \rightarrow (-1, -1, -1)$ and $(right, top, -far) \rightarrow (1, 1, 1)$:

$$\begin{pmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & \frac{-2(far \cdot near)}{far-near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Texture Mapping

Barycentric Coordinates: coordinate system for triangles based on linearly interpolating values at vertices:

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

α, β can be solved via

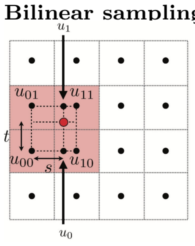
$$\frac{-(x - x_B)(y_C - y_B) + (y - y_B)(x_C - x_B)}{-(x_A - x_B)(y_C - y_B) + (y_A - y_B)(x_C - x_B)},$$
$$\frac{-(x - x_C)(y_A - y_C) + (y - y_C)(x_A - x_C)}{-(x_B - x_C)(y_A - y_C) + (y_B - y - C)(x_A - x_C)}$$

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

Texture mapping: applying 2D textures to 3D world space surfaces by sampling the texture space (texels) with Barycentric interpolation. *Magnification*: Each screen pixel is a small part of the texel. *Minification*: each pixel includes many texels.

Both cases lead to aliasing; resolve magnification with nearest, bilinear, or trilinear sampling.

Bilinear sampling:



Interpolated result given by $\text{lerp}(t, u_0, u_1)$ where $u_0 = \text{lerp}(s, u_{00}, u_{10})$, $u_1 = \text{lerp}(s, u_{01}, u_{11})$, and $\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$.

Resolve minification with **Mipmaps**: store an image pyramid of successively lower-resolutions; use resolution that matches screen sampling rate. Computing mipmap level:

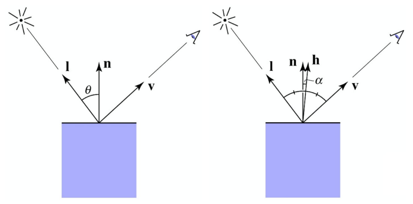
$$D = \log_2 L, \text{ where } L = \max\left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right)$$

In **trilinear sampling**, perform bilinear interpolation at level D and $D + 1$, then lerp based on fractional mipmap level.

Note: mipmapping is *not* anisotropic - it may favor one direction over the other.

Local shading

Assume a viewing direction $\mathbf{v}$, surface normal $\mathbf{n}$, and light direction $\mathbf{l}$.

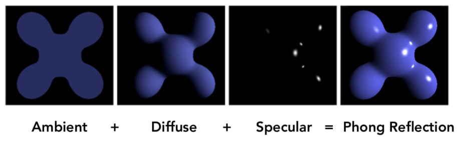


Lambertian/diffuse shading (left): $L_d = k_d(I/r^2)\max(0, \mathbf{n} \cdot \mathbf{l})$, where k_d is the diffuse coefficient and I/r^2 is the illumination from the source.

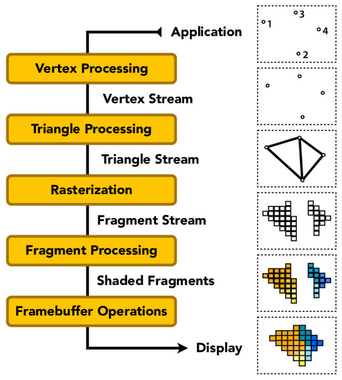
Specular shading (right): $L_s = k_s(I/r^2)\max(0, \mathbf{n} \cdot \mathbf{h})^p$ where k_s is the specular coefficient and $\mathbf{h} = \frac{\mathbf{v}+\mathbf{l}}{\|\mathbf{v}+\mathbf{l}\|}$.

Ambient shading: $L_a = k_a I_a$ (adds constant color to account for disregarded illumination).

The Blinn-Phong reflection model uses $L = L_a + L_d + L_s$.



Rasterization Pipeline



Curves

Cubic Hermite spline: Given $P_0 = h_0, P_1 = h_1, P'_0 = h_2, P'_1 = h_3$ (i.e. two points and their derivatives), construct a smooth cubic polynomial

$$P(t) = at^3 + bt^2 + ct + d$$

$$= \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}^\top \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{pmatrix}$$

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

where

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

are called the Hermite basis functions.
Catmull-Rom Spline: Given P_0, P_1, P_2, P_3, use cubic hermite spline with $h_0 = P_1, h_1 = P_2, h_2 = \frac{1}{2}(P_2 - P_0), h_3 = \frac{1}{2}(P_3 - P_1)$. The endpoints serve only to match the slopes between the previous and next values.

Bézier Curves

A cubic bézier curve is defined by 4 points; the curve starts at P_0, being tangent to the line between P_0 and P_1, and ends at P_3, being tangent to the line between P_2 and P_3.
1D **de Casteljau subdivision**: Given n points points $P_0, \dots, P_n$, compute a new set of $n-1$ points $P'_0, \dots, P'_{n-1}$ by

$$P'_i = (1-t)P_i + tP_{i+1}$$

Repeat until there is only one point left; the path traced out by this point for $0 \le t \le 1$ defines the Bézier curve.
Analytic solution: for n control points P_i, $\mathbf{P}(t) = \sum_{i=0}^{n} B_i(t)P_i$ where

$$B_i(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

are called the Bernstein basis polynomials. The cubic solution is $\mathbf{P}(t) = (1-t^3)P_0 + 3t(1-t)^2P_1 + 3t^2(1-t)P_2 + t^3P_3$, or in matrix form

$$\begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}^\top \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

Bézier surfaces: Given 4×4 control points, output a 2D surface parametrized by $(u, v) \in [0,1]^2$. One evaluation technique is to apply de Casteljau's algorithm with parameter u to each row of control points; then apply de Casteljau's algorithm with parameter v to those resulting points.

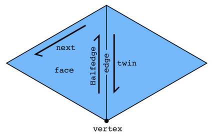
Geometry

Implicit geometry: Surface defined where $f(x, y, z) = 0$. Good for pathtracing.
Explicit geometry: All points given directly, $f : \mathbb{R}^2 \to \mathbb{R}^3; (u, v) \to (x, y, z)$. Good for rasterization.

The aspect of a surface which is unaffected by deformation is the **topology** of the surface (e.g. egg and a sphere have the same topology). A surface's **geometry** consists of those properties which do change when the surface is deformed (e.g. curvature, area, distance, angle).
Topological validity: A 2D manifold is a surface that when cut with a infinitesimally small sphere always yields a (curved) disk (and only one). Mesh manifolds always have the properties: edge connects two faces and two vertices, face consists of ring of edges and vertices, vertex consists of ring of edges and faces. $F - E + V = 2$ holds for a surface topologically equivalent to a sphere.

The **half-edge** structure is a common topological data structure for triangle meshes.

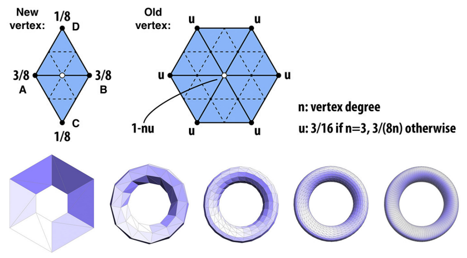
```
struct Halfedge {
  Halfedge *twin;
  Halfedge *next;
  Vertex *vertex;
  Edge *edge;
  Face *face;
}
```



Each vertex, face, and edge points to one its half-edges.
Example to traverse all edges around a vertex:

```
  Halfedge* h = v->halfedge;
  do {
    process(h->edge);
    h = h->twin->next;
  }
  while( h != v->halfedge );
```

Local mesh editing operations include splitting, flipping, and collapsing edges. Global mesh editing operations include subdivision, downsampling, and regularization.
Loop subdivision: Split every edge of original mesh in any order; flip any edges that touch a new and old vertex; update vertex positions by



Catmull-Clark subdivision: Starting with a mesh of arbitrary polyhedron: add a face point to the centroid of each face; add an edge point (average of 2 neighboring vertices and 2 face points) to each edge; connect each face point to each edge point; update each *original* vertex position P to

$$P_{\text{new}} = \frac{F + 2R + (n-3)P}{n}$$

where F is the average of adjacent face points, R is the average of (original) adjacent edge midpoints, and n is the degree/valence of the vertex. The result is a smoother mesh composed only of quadrilaterals.

Radiometry/Photometry

Photometry is the measurement of light in terms of its perceived brightness to the human eye, while radiometry is the measurement of light in terms of absolute power.

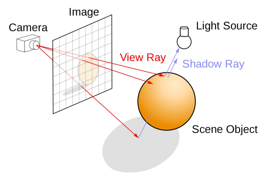
Radiometry	Units	Photometry	Units
Radiant Energy	Joules (W·sec)	Luminous Energy	Lumen·sec
Radiant Power	W	Luminous Power	Lumen (Candela sr)
Radiant Intensity	W/sr	Luminous Intensity	Candela (Lumen/sr)
Irradiance (in) Radiosity (out)	W/m²	Illuminance (in) Luminosity (out)	Lux (Lumen/m²)
Radiance	W/m²/sr	Luminance	Nit (Candela/m²)

Solid angle: $\Omega = A/r^2$; a sphere has 4π steradians.
In principle, irradiance is calculated by integrating radiance over a solid angle. In practice, Monte Carlo integration is used; it avoid the curse of dimensionalty and error (σ) falls as $1/\sqrt{N}$ for N samples.

$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}, \quad X_i \sim p(x)$$

It can be slow to converge; can improve by using non-uniform sampling PDF to favor directions where incoming radiance will be non-zero (e.g. importance sampling in direct lighting).

Ray Tracing



Ray: $\mathbf{r}(t) = \mathbf{O} + t\mathbf{D}, \quad 0 \le t < \infty$
Triangle intersection: Given a triangle with vertices $\mathbf{P_0}, \mathbf{P_1}, \mathbf{P_2}$, we set a point $\mathbf{P}$ inside the triangle $\mathbf{P} = (1 - b_1 - b_2)\mathbf{P_0} + b_1\mathbf{P_1} + b_2\mathbf{P_2}$ and set $\mathbf{r}(t) = \mathbf{P}$ to solve for t, b_1, b_2.
Möller–Trumbore:

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

where

$$\mathbf{S} = \mathbf{O} - \mathbf{P_0}$$
$$\mathbf{E_1} = \mathbf{P_1} - \mathbf{P_0} \qquad \mathbf{S_1} = \mathbf{D} \times \mathbf{E_2}$$
$$\mathbf{E_2} = \mathbf{P_2} - \mathbf{P_0} \qquad \mathbf{S_2} = \mathbf{S} \times \mathbf{E_1}$$

Sphere intersection: A sphere centered at $\mathbf{C}$ with radius R is given by the set of points $\mathbf{P}$ that satisfy $(\mathbf{P} - \mathbf{C})^\top(\mathbf{P} - \mathbf{C}) - R^2 = 0$. The intersection solution is $at^2 + bt + c = 0$ where

$$a = \mathbf{D} \cdot \mathbf{D}$$
$$b = 2(\mathbf{O} - \mathbf{C}) \cdot \mathbf{D}$$
$$c = (\mathbf{O} - \mathbf{C}) \cdot (\mathbf{O} - \mathbf{C}) - R^2$$

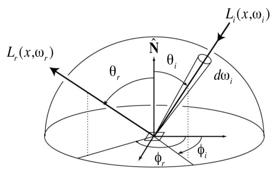
Recover the value(s) of t with the quadratic formula, taking the closest solution.
Ray-plane intersection: if $\mathbf{P'}$ is a point on the x-plane, then the intersection occurs at

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

(with similar results for the y and z planes).
Accelerating intersections: use spatial partitioning data structure (e.g. Oct-Tree, KD-Tree, BSP-Tree) or object partioning schemes (e.g. BVHs) to reduce ray-primitive intersection complexity from linear to log.
In a BVH, internal nodes store a bounding volume and children pointers; leaf nodes store a list of primitives. Intersection psuedocode:

```
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;
```

Global Illumination



Reflectance equation:

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

Define the **transport function** $tr(p,\omega_i)$ to return the first intersection point in the scene along ray (p,ω).

Rendering equation:

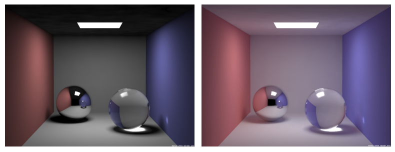
$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_{H^2} f_r(p,\omega_i \to \omega_o) L_o(tr(p,\omega_i),-\omega_i)\cos\theta_i d\omega_i$$

The radiance at intersection point `p` from direction ω is the sum of direct emission at `p` and `AtLeastOneBounceRadiance(p, `$-\omega$`)`:

```
AtLeastOneBounceRadiance(p, $\omega_o$)
  L = OneBounceRadiance(p, $\omega_o$);
  $\omega_i$, pdf = p.brdf.sampleDirection();
  p' = intersectScene(p, $\omega_i$);
  cpdf = continuationProb(p.brdf, $\omega_i$, $\omega_o$);
  if (random() < cpdf)
    L += AtLeastOneBounceRadiance(p', $-\omega_i$)
    * p.brdf($\omega_i$, $\omega_o$) * cos($\theta$) / pdf / cpdf;
  return L

OneBounceRadiance(p, $\omega_o$)
  return DirectLightingSampleLights(p, $\omega_o$);

DirectLightingSamplingLights(p, $\omega_o$)
  L, $\omega_i$, pdf = lights.sampleDirection(p);
  if (scene.shadowIntersection(p, $\omega_i$))
    return 0;
  else
    return L * p.brdf($\omega_i$, $\omega_o$)
    * cos($\theta$) / pdf;
```



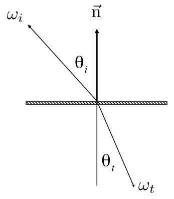
Material Modeling

Bidirectional Reflectance Distribution (BRDF): Function $f_r(\omega_i,\omega_r)$ of incoming light direction ω_i and outgoing direction ω_r that returns the ratio of reflected radiance exiting along ω_r to the irradiance incident on the surface from direction ω_i. It is non-negative and has the reciprocity principle.

Example kinds of reflection: **ideal specular** (mirror), **ideal diffuse** (equal in all directions), **glossy specular** (majority of light reflected toward mirror direction), or **retro-reflective** (light reflected back towards light source - like bike reflectors).

Lambert's cosine law: The radiant/luminous intensity observed from a diffusely reflecting surface is proportional to $\cos(\theta) = \mathbf{I}\cdot\mathbf{n}$, where θ is the angle between the normal $\mathbf{n}$ and the incident light source and $\mathbf{I}$ is a vector from the hit point to light source.

Refraction is described in physics by Snell's law:



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

where η_i and η_t are the indices of refraction for the incident and exiting ray, respectively - larger values mean more optically dense.

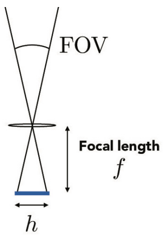
Isotropic materials have the same reflective properties in all directions; **anisotropic** materials reflect light preferentially along one direction.

A **microfacet** material is composed of many tiny facets (called microfacets), each of which is a perfect specular reflector. The BDRF is of the form

$$f_r(\omega_i,\omega_r) = \frac{F(\omega_i,h)G(\omega_i,\omega_o,h)D(h)}{4(n\cdot\omega_i)(n\cdot\omega_o)}$$

where h is the half vector, F is the fresnel term (reflection of incident light between different refractive indices), G is the shadow-masking term, D is the distribution of normals.

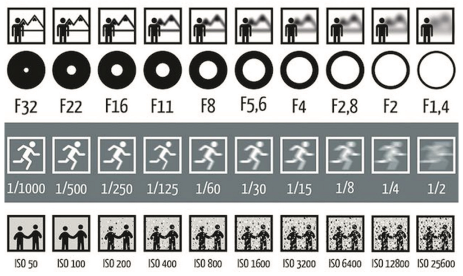
Cameras



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

Aperture (f-stop): f/D, for aperture diameter D. Irradiance proportional to square of D; inverse square of f. Increasing 2 f-stops doubles depth of field.

ISO (gain): sets sensitivity vs noise; linear effect (e.g. ISO 200 needs half as much light as ISO 100).



Depth of field (range of depths in focus) is controlled by aperture, focal length, object distance.

Exposure is controlled by aperture, shutter, ISO.

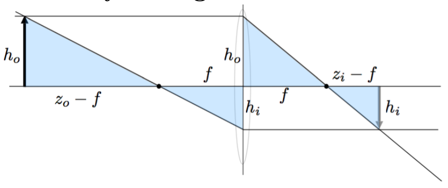
Motion blur is controlled by shutter speed.

A **thin lens** has a thickness that is negligible compared to the radii of curvature of the lens surfaces.

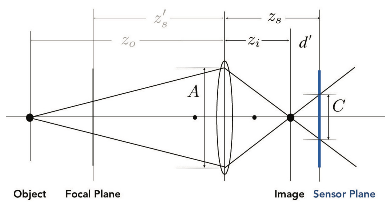
Thin lens equation: $\frac{1}{f} = \frac{1}{z_i} + \frac{1}{z_o}$ where z_i is the distance from the lens to the formed image and z_o is the distance to the object.

Magnification is given by $m = z_i/z_o$.

Gauss' ray-tracing construction:



Circle of confusion: an optical spot caused by a cone of light rays from a lens not coming to a perfect focus.



$$C = A\left(\frac{d'}{z_i}\right) = A\frac{|z_s - z_i|}{z_i}$$

Image Sensors

In a CCD image sensor, pixels are represented by p-doped metal-oxide-semiconductors (MOS) capacitors. They allow the conversion of incoming photons into electron charges at the semiconductor-oxide interface; the CCD is then used to read out these charges.

Quantum efficiency of device: $\frac{\#\text{ electrons}}{\#\text{ photons}}$

Color filter arrays often have more green to reflect human perception (e.g. Bayer filter, grid of RGGB).

Demosaicking algorithms use sophisticated interpolation techniques to produce RGB image from raw color filter data.

The **pixel fill factor** is the fraction of photosensitive pixel area (rest is circuitry). Improved with per-pixel *microlenses* that focus light onto the photosensitive portion.

FSI vs BSI: Back-side illuminated puts metal below diode below lens, whereas front-side illuminated puts metal in-between lens and diode. BSI makes more light hit the diode.

Imperfect fill factors and color subsampling result in aliasing; all cameras have an **antialiasing filter** that splits each ray over 2x2 pixels.

The number of photons arriving at a pixel follows a Poisson distribution.

Signal-to-noise (SNR): $\frac{\mu}{\sigma} = \frac{\lambda}{\sqrt{\lambda}} = \sqrt{\lambda}$

Animation

Common animation principles:

1. Squash and stretch
2. Anticipation
3. Staging
4. Straight ahead and pose-to-pose
5. Follow through
6. Ease-in and ease-out
7. Arcs
8. Secondary action
9. Timing
10. Exaggeration
11. Solid drawings
12. Appeal

In keyframe animation, each frame is a vector of parameter values; computer create in-between frames ("**tweens**") by interpolating these values.

Forward kinematics: Artist-specified movement of skeleton. A skeleton has a topology, geometric relations, and a tree structure; 3 joint types: pin (1D rotation), ball (2D rotation), and prismatic joint (translation). Computer determines position of end-effector.

Inverse kinematics: Artist specifies position of end-effector, computer solves for appropriate joint angles. Difficult optimization problem; often use data-driven approaches (e.g. motion capture) to get natural-looking motion.

Mass-spring systems are commonly used to model deformable meshes. By Hooke's law, for a spring with rest length l and endpoints at $\mathbf{a}$ and $\mathbf{b}$,

$$F_{a\to b} = k_s \frac{\mathbf{b}-\mathbf{a}}{\|\mathbf{b}-\mathbf{a}\|}(\|\mathbf{b}-\mathbf{a}\| - l)$$

where k_s is the spring constant. In practice, we apply damping (force proportional to velocity in opposing direction) as friction.
Dynamical simulation require **integration** of equations of motion.
Finite difference methods (e.g. Euler's method) always lead to compounding errors (instability), no matter the step size.
Modified Euler:
$x_{t+dt} = x_t + \frac{dt}{2}(v_t + v_{t+dt})$
$v_{t+dt} = v_t + a_t dt$
Verlet integration:
$x_{t+dt} = x_t + v_t dt + a_t dt^2$
$v_t dt = x_t - x_{t-dt}$
Adaptive step size: Compute x_T, Euler step of size T. Computer $x_{T/2}$, two Euler steps of size $T/2$. Compute error, check if greater than threshold, reduce step size if necessary.

Color

Tristimulus Theory of Color: In human visual system, any color spectra is perceived by 3 cones with different response curves.
Metamers: If two spectra produce the same tristimulus values, they are visually indistinguishable; this is critical to modern color reproduction.
Color perceived from a display spectra with values R, G, B:

$$\begin{bmatrix} S \\ M \\ L \end{bmatrix}_{\text{disp}} = \begin{bmatrix} — & r_S & — \\ — & r_M & — \\ — & r_L & — \end{bmatrix} \begin{bmatrix} | & | & | \\ s_R & s_G & s_B \\ | & | & | \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

where s_R, s_G, s_B are the emission spectra of the display and r_S, r_M, r_L are the spectral response values.
Color percieved for real scene spectra, s:

$$\begin{bmatrix} S \\ M \\ L \end{bmatrix}_{\text{real}} = \begin{bmatrix} — & r_S & — \\ — & r_M & — \\ — & r_L & — \end{bmatrix} \begin{bmatrix} | \\ s \\ | \end{bmatrix}$$

Reproduce s by setting these equations equal and solving for R, G, B.
The **gamut** is a subset of colors that can be accurately represented. A chromaticity diagram (CIE 1931 shown below) is a projection of the 3D LMS curves onto the xy plane. The triangle represents the **sRGB** gamut, which is relatively limited but widely-used.

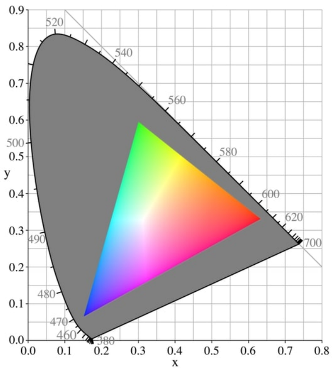


Image Processing

JPEG compression: Convert image to Y'CbCr color space (luma, blue-yellow and red-green). Downsample chroma channels, luma stays same. Uses discrete cosine transform (DCT) on each 8x8 block to quantize and encode values.
Sobel edge detection: Find pixels with large gradients $G = \sqrt{G_x^2 + G_y^2}$.
Horizontal gradients Vertical gradients

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Data-dependent filters like median filter or bilateral filter (preserves edges) are useful for noise removal.

GPUs and performance

A **GPU** is a heterogeneous multi-processor chip with specialized hardware for shading, textures, rasterization, video decoding, etc. Relative to CPUs, GPUs have 1) more cores; extranneous hardware is removed to support lots of cores, 2) simultaneous instruction streams; amortize cost/complexity of managing an instruction stream across many ALUs with SIMD processing, 3) interleave processing of many fragments on a single core to avoid stalls caused by high latency operations.

The big idea is emphasis on *aggregate throughput* rather than individual latency.

Misc

Fourier transform:
$F(\omega) = \int_{-\infty}^{\infty} f(x)e^{-2\pi i\omega x}dx$
Inverse transform:
$f(x) = \int_{-\infty}^{\infty} F(\omega)e^{2\pi i\omega x}d\omega$
Euler's formula: $e^{ix} = \cos x + i\sin x$
Convolution:
$(f * g)(x,y) = \sum_{i,j=-\infty}^{\infty} f(i,j)I(x-i, y-j)$
$\mathbf{E}[X+Y] = \mathbf{E}[X] + \mathbf{E}[Y]$
$\mathbf{E}[aX] = a\mathbf{E}[X]$
$\text{Var(X)} = \mathbf{E}[X - \mathbf{E}[X]^2] = \mathbf{E}[X^2] - \mathbf{E}[X]^2$
$\text{Var(aX)} = a^2\text{Var(X)}$
Equation of **ellipsoid** centered at (x_0, y_0, z_0) with principal axis lengths (a, b, c):

$$\left(\frac{x-x_0}{a}\right)^2 + \left(\frac{y-y_0}{b}\right)^2 + \left(\frac{z-z_0}{c}\right)^2 = 1$$

Equation of a **plane**:

$$\mathbf{n}\cdot(\mathbf{r} - \mathbf{r}_0) = 0$$

where $\mathbf{n}$ is normal to the plane and $\mathbf{r}_0$ is a point on the plane.
The **CDF** F_X of a continuous random variable X with PDF f_X:

$$F_X(x) = \int_{-\infty}^{x} f_X(t)\,dt$$

Inversion method: given a uniform random variable $U \in [0,1]$, we can generate a random variable X from any other one dimensional distribution by returning $X = F^{-1}(U)$.