COS426 Exam Revision Guide

Image

Definition: image is a 2D rectilinear array of pixels

Pixel: sample of a function at a position; for a photo, each pixel is a function of radiance arriving at a sensor

Plenoptic Function the 7D plenoptic function that describes the radiance arriving at any position (x,y,z), in any direction (θ, ϕ), at any time (t), at any frequency (λ)

Electromagnetic Spectrum visible light from red (4.3e14 Hz, 700nm) to violet (7.5e14 Hz, 400nm)

Color characterized by its spectrum, which is the magnitude of energy at every visible frequency

RGB due to each of the three types of cones on the human retina (tristimulus theory of color); colors are **additive**

CMY Color Model is useful for printers because colors are subtractive; (Cyan, Yellow, Magenta)

HSV Color Model is hue (around the cone), value (along the slope), saturation (from inside out, radius); useful for user interfaces because dimensions are intuitive

- Hue = dominant frequency (highest peak in color spectrum)
- saturation = excitation purity (ratio of higest to rest)
- value = luminance (area under curve)

XYZ Color Model (CIE) derived from perceptual experiments; all spectra that map to same XYZ give same visual sensation; useful in reasoning about color gamut coverage, identifying complementary colors, and determining dominant wavelength and purity

La*b* color model is a non-linear compression of XYZ color space based on perception, useful for measuring perceptual differences between colors

LMS color space represents the responses of the three difference types of cones in out eyes

Image Processing

Luminance measures perceived "gray-level" of pixel; $L = 0.30 * R + 0.59 * G + 0.11 * B$

To adjust **brightness**, you can (1) convert to HSL, scale L, convert back; or (2) scale R, G, B directly

To adjust **contrast**, you compute mean luminance L over the hwole image and then scale deviation from L for each pixel;

Histogram Equalization changes distribution of luminance values to cover full range [0-1]

Convolution outputs a weighted sum of values in neighborhood of input image; pattern of weights is called the filter or kernel; e.g. Edge detection with

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Sharpen with

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

You can separate some filters (like Gaussian) to speed up performance

Bilateral Filter combines Gaussian filtering in both spatial and color domains

Quantization reduces intensity resolution due to the constraints that frame buffers have limited number of bits per pixel; physical devices have limited dynamic range;

Dithering distributes errors among pixels, hence reduces visual artifacts due to quantization, by exploiting spatial integration in our eye; random dither versus ordered dither versus error diffusion dither (spread quantization error over neighbor pixels);

Sampling, Resampling, and Warping

about **Transformation** and **Combining Images**

Transformation:

1. specify where every pixel goes (mapping)
2. compute colors at destination pixels (resampling)

Naive resampling can cause visual artifacts (e.g. when scaling up/down), which is known as **aliasing**, due to under-sampling; when high frequencies masquerade as low ones;

A signal can be reconstructed form its samples, *iff* the original signal has no content $\geq$ 1/2 the sampling frequency, as proposed by Shannon. The minimum sampling rate for a "*bandlimited*" function is called the **Nyquist rate**. A signal is *bandlimited* if its highest frequency is bounded. the frequency is called the bandwidth.

Forward mapping (requires separate butter to store weights) versus **reverse mapping** (requires inverse of mapping function, random access to original image); reverse mapping is usually preferable;

Combining images:

1. segmentation of image into layers/regions
    1. draw matte for every frame
    2. graph-cut (draw a few strokes to separate image regions along minimal cuts)
    3. flash matting
2. blend into single image seamlessly
    1. alpha channel controls the linear interpolation of foreground and background pixels when elements are composited; $\alpha = 0$ means transparent, or no pixel coverage; $\alpha =$1 means opage, or full pixel coverage;
    2. **Beier & Neeley** uses pairs of lines to specify warp;

3D Modeling

Scene is usually approximated by 3D primitives such as point (3 coordinates, infinitely small), vector (direction and magnitude, no location), line segment (linear path between 2 points), ray (line segment with one endpoint at infinity), line (line segment with both endpoints at infinity), plane (linear combination of 3 points), polygon (set of points "inside" a seqeuence of coplanar points in counter-clockwise order).

dot product: $v_1 \cdot v_2 = ||v_1|| * ||v_2||cos(\theta)$; cross product: vector perpendicular to both v1 and v2: $||v_1 \times v_2|| = ||v_1|| * ||v_2||sin(\theta)$

3D Object Representation: analogous to Turing-equivalence, each representation has enough expressive power to model the shape of any geometric object; it is possible to perform all geometric operations with any fundamental representation

1. Point
    1. **Range Image**: under points, set of 3D points mapping to pixels of depth image. can be acquired from range scanner
    2. **Point Cloud**: unstructured set of 3D point samples; acquired from range finder, computer vision, etc
2. Surface
    1. **Polygonal Mesh**: using surface; connected set of polygons (often triangles)
    2. **Subdivision Surface**: coarse piecewise linear mesh & subdivision rule; smooth surface is limit of sequence of refinements; The smooth surface can be calculated from the coarse mesh as the limit of a recursive process of subdividing each polygonal face into smaller faces that better approximate the smooth surface.
    3. **Parametric Surface**: tensor-product spline patches where each patch is parametric function with careful constraints to maintain continuity;
    4. **Implicit Surface**: set of all points satisfying F(x,y,z)=0;
3. Solid
    1. **Voxel grid**: uniform volumetric grid of samples; often acquired via simulation or from CAT, MRI, etc
    2. **BSP Tree**: hierarchical binary space partition with solid/empty cells labeled;

    constructed from polygonal representations
  3. **CSG**: constructive solid geometry; set of operations (union, difference, intersection) applied to simple shapes
  4. **Sweep**: solid swept by curve along trajectory;

4. High-level structures

  1. **Scene Graph**: union of objects at leaf nodes;
  2. **Application Specific**:

Efficiency vs. Simplicity vs. Usability

3D Polygonal Mesh: set of polygons representing a 2D surface embedded in 3D; most common surface representation; fast rendering; must consider irregular vertex sampling, must handle/avoid topological degeneracies; not so accurate or concise or intuitive or with guaranteed validity/smoothness;

Representation of 3D polygonal mesh: (preferably, fast traversal, small memory requirement, fast update)

- independent faces: each face lists vertex coordinates (redundant vertices, no adjacency information)
- vertex and face tables: each face lists vertex references; knows shared vertices, still no adjacency information
- adjacency lists: store all vertex, edge, and face adjacencies
  - efficient adjacency traversal
  - extra storage
- Winged edge: adjacency encoded in edges; all adjacencies in O(1) time; little extra storage (fixed records), but arbitrary polygons;
- **half edge**: adjacency encoded in edges; similar to winged-edge, except adjacency encoded in half-edges;

Parametric Curves: defined by parametric functions; use functions that "blend" control points;

Parametric **polynomial** curves: polynomial blending functions; easy to compute, **infinitely continuous**, easy to derive curve properties;

Piecewise Parametric Polynomial Curves: **Splines**: split curve into segments; each segment defined by low-order polynomial blending subset of control vertices; provide *local control* and efficiency;

Cubic B-Splines: local control; C^2 continuity at joints (infinitely continuous within each piece); approximating; convex hull; constraints are implied by C^2 continuity; e.g. 4 cubic polynomials for 4 vertices, with 16 variables in total, and 15 constraints (and 1 more convenient constraint);

Cubic Bezier: local control, continuity depends on control points, interpolating (every third for cubic); curve is contained within convex hull of control polygon; neighboring segments shared **1** control point

Parametric Surfaces: defined by parametric functions which define mapping from (u,v) to (x,y,z); to model arbitrary shapes, surface is partitioned into **parametric patches**; each patch is defined by blending control points (similar to parametric curves), and Point Q(u,v) on any patch is defined by combining control points with polynomial blending functions. (B-Spline patch has B-Spline matrix, Bezier patch has Bezier matrix, etc). Bezier patch is made form interpolating 4 corner points; convex hull; has local control;

Piecewise Polynomial Parametric Surfaces (**B-Spline Surface**) must maintain continuity across seams (similar to parametric splines).

For **Bezier Surfaces**, continuity constraints are similar to the ones for Bezier splints; C^0 continuity requires aligning boundary curves; C^1, additionally requires aligning boundary curve derivatives;

Parametric Surfaces are guaranteed smoothness, intuitive, concise, accurate, natural, but not easy to acquire or intersect, no guaranteed validity, cannot apply to arbitrary topology.

Subdivision Surfaces: guaranteed continuity with repeated application of topology refinement (splitting faces) and geometry refinement (weighted averaging); base mesh + limit surface.

Subdivision Surface Summary:

- Advantages: Simple method for describing complex surfaces; Relatively easy to

implement; Arbitrary topology; Intuitive specification; Local support; Guaranteed continuity;Multiresolution
- Difficulties:;Parameterization; Intersections

Continuity: a curve/surface with G^k continuity has a continuous $k-$th derivative, geometrically.

Linear subdivision: quad mesh, finding midpoints + averaging vertex positions;

Catmull-Clark subdivision: (assignment) one round of subdivision produces all quads; C^2 almost everywhere, C^1 at vertices with valence $\neq$ 4; doesn't interpolate input vertices; within convex hull; approximating;

Loop Subdivision: on pure triangle meshes; linear subdivision + averaging rules for "even/odd" (white/black) vertices with different weights; C^2 almost everywhere, C^1 at vertices with valence $\neq$ 6; doesn't interpolate input vertices; within convex hull; approximating;

Comparison

- Parametric surfaces
    - Provide parameterization
    - More restriction on topology of control mesh
    - Some require careful placement of control mesh vertices to guarantee continuity (e.g., Bezier)
- Subdivision surfaces
    - No parameterization
    - Subdivision rules can be defined for arbitrary topologies
    - Provable continuity for all placements of control mesh vertices

Implicit Surfaces: represent surface with function over all space; and the surface is defined implicitly by function, (e.g. f(x,y,z) = 0 is on the surface, < 0 is inside the surface, > 0 is outside the surface); normals are defined by partial derivatives (Normal(x,y,z) = normalize(df/dx, df/dy, df/dz)); hence it's efficient to check if point is inside - just evaluate it; it's efficient to find intersections, just substitute; efficient for boolean operations (CSG); efficient for topology changes, as surface is not represented explicitly; (Parametric is efficient at *marching* along surface & rendering)

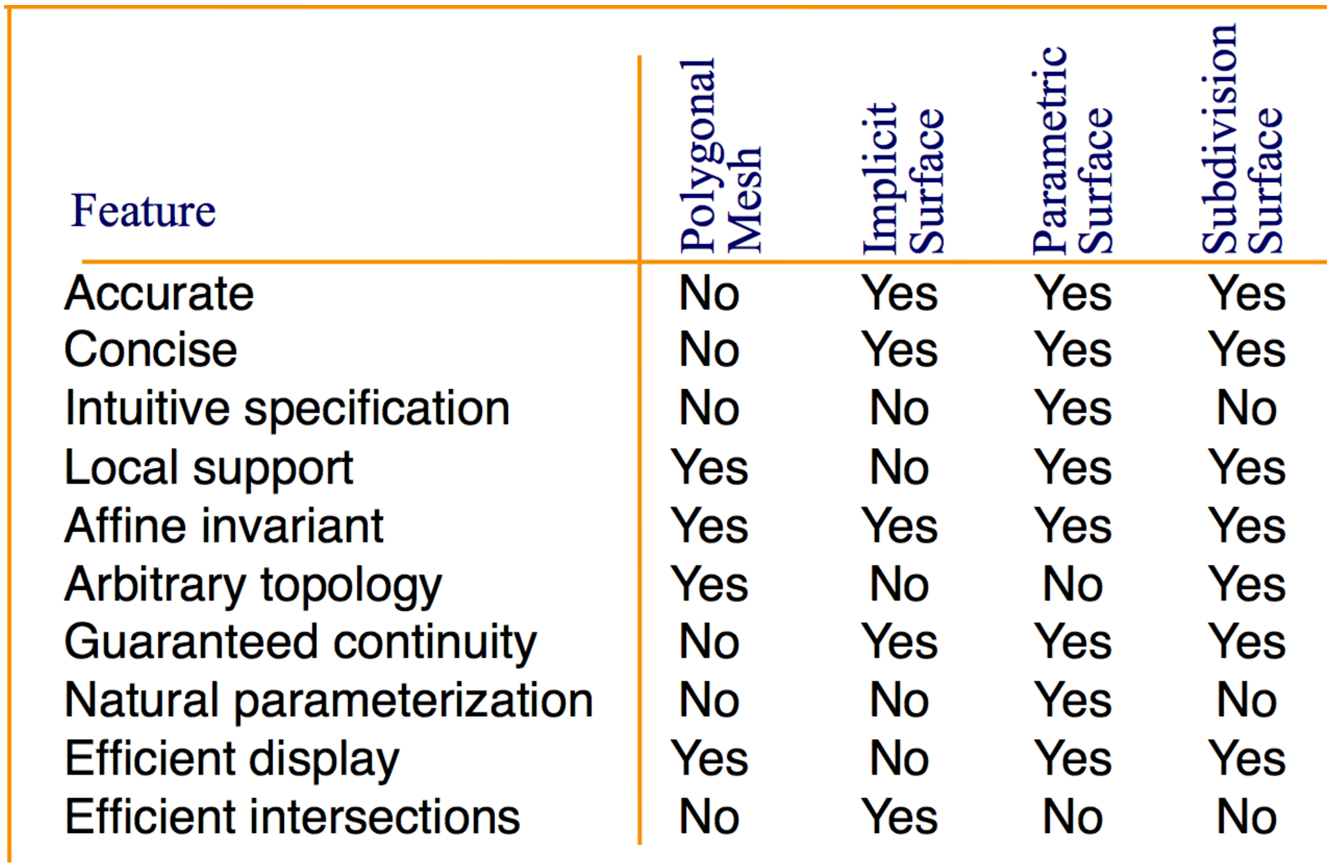
Algebraic Surfaces: implicit surface, defined by polynomials; has an equivalent parametric surface: Tensor product patch of degree m and n curves yields algebraic function with degree 2mn; Intersection of degree m and n algebraic surfaces yields curve with degree mn. Function extends to infinity hence must trim to get desired patch.

Voxels are samples from regular array of 3D samples (like image), called voxels due to "volume pixels"; $O(n^3)$ storage for cubes;

Basis functions: whose implicit function is sum of basis functions; e.g. Blobby Models whose implicit function is sum of Gaussians;

Implicit Surface Summary:

- Advantages: Easy to test if point is on surface; Easy to compute intersections/unions/differences; Easy to handle topological changes
- Disadvantages: Indirect specification of surface; Hard to describe sharp features; Hard to enumerate points on surface -> slow rendering;



Feature	Polygonal Mesh	Implicit Surface	Parametric Surface	Subdivision Surface
Accurate	No	Yes	Yes	Yes
Concise	No	Yes	Yes	Yes
Intuitive specification	No	No	Yes	No
Local support	Yes	No	Yes	Yes
Affine invariant	Yes	Yes	Yes	Yes
Arbitrary topology	Yes	No	No	Yes
Guaranteed continuity	No	Yes	Yes	Yes
Natural parameterization	No	No	Yes	No
Efficient display	Yes	No	Yes	Yes
Efficient intersections	No	Yes	No	No

Solid Modeling: represent solid interiors of objects;

Use of **Voxels**:

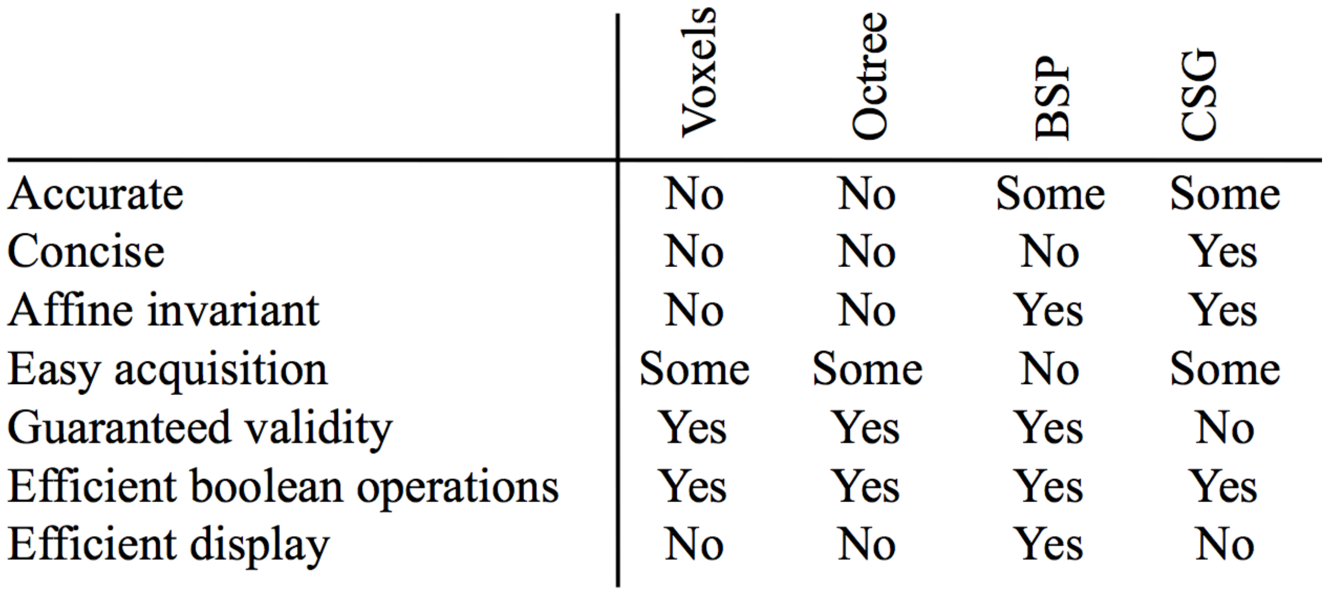
- +:
    - simple, intuitive, unambiguous;
    - same complexity for all objects
    - natural acquisition for some applications;
    - trivial boolean operations;
- -:
    - approximate
    - not affine invariant
    - expensive display
    - large storage requirements

Voxel resolution -> by quadtrees (amortized O(1)) & octrees to refine resolution of voxels hierarchically; hence more concise and efficient for non-uniform objects

BSP Tree: visibility ordering; hierarchy of convex regions (useful for collision);

Constructive Solid Geometry (CSG): represent solid objects as hierarchy of boolean operations; union, intersection, difference; you can create a new CSG node by joining subtrees;

Sweeps: swept volume; sweep one curve along path of another curve;



	Voxels	Octree	BSP	CSG
Accurate	No	No	Some	Some
Concise	No	No	No	Yes
Affine invariant	No	No	Yes	Yes
Easy acquisition	Some	Some	No	Some
Guaranteed validity	Yes	Yes	Yes	No
Efficient boolean operations	Yes	Yes	Yes	Yes
Efficient display	No	No	Yes	No

Scene Graph: hierarchy (DAG) of nodes where each may have: geometry representation, modeling transformation, parents and/or children, **bounding volume**;

- Advantage:
  - Allows definitions of objects in own coordinate systems
  - Allows use of object definition multiple times in a scene
  - Allows hierarchical processing (e.g., intersections)
  - Allows articulated animation

Transformations: scale, rotate, shear, translate in basic 2D;apply transformation to point by multiplying matrix by column vector; transformations can be combined by multiplication; but only *linear* 2D transformations can be representation with a 2x2 matrix, so no 2D translation matrix (2D translation is represented by a 3x3 matrix, with points represented with homogeneous coordinates).

2D Identity?

$$x' = x$$
$$y' = y$$

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

2D Scale around (0,0)?

$$x' = sx * x$$
$$y' = sy * y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Rotate around (0,0)?

$$x' = \cos\Theta * x - \sin\Theta * y$$
$$y' = \sin\Theta * x + \cos\Theta * y$$

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

2D Shear?

$$x' = x + shx * y$$
$$y' = shy * x + y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & shx \\ shy & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Mirror over Y axis?

$$x' = -x$$
$$y' = y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Mirror over (0,0)?

$$x' = -x$$
$$y' = -y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

For 2D translation:

$$x' = x + tx$$
$$y' = y + ty$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

You can convert 2D transformations to 3x3 matrices by adding homogeneous coordinates;

Affine Transformations are combinations of linear transformations and translations; Origin does not necessarily map to origin.

Projective Transformations: represent camera projection in same framework as modeling transformations; Origin does not necessarily map to origin; Point at infinity may map to finite point; Parallel lines do not necessarily remain parallel; Ratios are not preserved

Matrix composition is associative but not commutative (order matters);

3D transformations can use homogenous coordinates of (x,y,z,w) and 4x4 transformation matrices.

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Identity

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Mirror over X axis

Rotate around Z axis:

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

Rotate around Y axis:

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

Rotate around X axis:

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

3D Rendering

constructing 2D images from 3D models. Rendering is a problem in sampling and reconstruction as scene can be sampled with any ray

Camera Models: the most common model is a pin-hold camera, where light rays arrive along paths toward focal point and no lens effects are present

Ray Casting: for each sample, Construct ray from eye position through view plane, Find first surface intersected by ray through pixel and then Compute color of sample based on surface radiance.

Lighting Simulation: light source emission + surface reflectance + atmospheric attenuation + camera response

Shadows: occlusions from light sources, accounting for soft shadows with area light source;

Ray Casting: The color of each pixel on the view plane depends on the **radiance** emanating along rays from visible surfaces in scene

1. generate rays
2. intersection tests
3. lighting calculations

Ray-Sphere Intersection II: trivial checks;

Ray-Triangle: barycentric coordinates;

Acceleration tricks:

1. bounding volumes: if ray doesn't intersect bounding volume, the it can't intersect its contents; If already found a primitive intersection closer than intersection with bounding box, then skip checking contents of bounding box; Sort child bounding volume intersections and then visit child nodes in front-to-back order

2. uniform grid: construct uniform grid over scene; index primitives according to overlaps with grid cells; fast, incremental; only check primitives in intersected grid cells; tricky to pick the right resolution;

3. Octree: construct adaptive grid over scene; recursively subdivide box=shaped cells into 8 octants, index primitives by overlaps with cells; trade-off fewer cells for more expensive traversal;

    1. or, check rays versus octree boxes hierarchically: compute octree boxes while descending tree, sort eight boxes front-to-back at each level, check primitives/children inside box

4. Binary Space Partition Tree (BSP): recursively partition space by planes; BSP tree nodes store partition plane and set of polygons lying on that partition plane; every part of every polygon lies on a partition plane. Then traverse nodes of BSP tree front-to-back, visit halfspace (child node) containing P_0, intersect polygons lying on partition plane, visit halfspace (other child node) not containing P_0;
5. parallelism
6. memory coherence for large scenes
7. screen space coherence - check more than one ray at once

• General concepts:

Sort objects spatially!

Make trivial rejections quick!

Perform checks hierarchically!

Utilize coherence when possible

Expected time is sub-linear in number of primitives

Point Light Source: models omni-directional point source, attenuation with distance:

$$I_L = \frac{I_0}{c_a + l_a d + q_a d^2}$$

where I_0 is intensity, d is distance, c_a, I_a, q_a are coefficients; based on inverse square law;

Directional Light Source: models point light source at infinity; no attenuation with distance;

$$I_L = I_0$$

Spot Light Source: models point light source with direction, with falloff (a range of angles that light shines on) and cutoff;

Scattering at Surfaces: bidirectional reflectance distribution function that describes the aggregate fraction of incident energy, with arriving direction and leaving direction

- Empirical Models: ideally measure BRDF for "all" combinations of angles; difficult in practice and too much storage
- Parametric Models: approximate BRDF with simple parametric function that is fast to computer, e.g. Phong

OenGL Reflectance Model: based on Phong model;

- diffuse reflection + : assume surface reflects equally in all directions; brightness of surface depends on angle of incident light; Lambertian model is $I_D = K_D(N \cdot L)I_L$
- specular reflection + : reflection is strongest near mirror angle, depends on angle of incident light and angle to viewer; Phong model contains the term $(\cos\alpha)^n$; the formula is $I_S = K_S(V \cdot R)^n I_L$
- emission + : represents light emanating directly from surface;
- ambient: represents reflection of all indirect illumination (this is hack)

Single light source: $I = I_E + K_A I_{AL} + K_D(N \cdot L)I_L + K_S(V \cdot R)^n I_L$

can add a shadow term to tell if light sources are blocked;

Recursive Ray Tracing (global illumination): also trace secondary rays from hit surfaces *add formula in cheatsheet*; shadows, mirror reflection, pure refraction

Snell's Law;

Illumination Terminology

1. **Radiant power** [flux] (Φ): Rate at which light energy is transmitted (in Watts).
2. **Radiant Intensity** (I): Power radiated onto a unit solid angle in direction (in Watts/sr)» e.g.: energy distribution of a light source (inverse square law)
3. **Radiance** (L): Radiant intensity per unit projected surface area (in Watts/m2sr)» e.g.: light carried by a single ray (no inverse square law)•
4. **Irradiance** (E): Incident flux density on a locally planar area (in Watts/m2)» e.g.: light hitting a surface at a point
5. **Radiosity** (B): Exitant flux density from a locally planar area (in Watts/m2)

Shadows:

- soft shadows: from area light sources; umbra = fully shadowed; penumbra = partially shadowed;

Radiance leaving point x on surface is sum of reflected **irradiance** arriving from other surfaces; radiance is power per unit area per unit solid angle, measured in W/m^2/sr;

Rendering Equation: computes radiance in outgoing direction by integrating reflections over all incoming directions;
$L_0(x', w') = L_e(x', w') + \int_\Omega f_r(x, w, w')(w \cdot n)L_i(x', w)dw$

- Ω is hemisphere
- n is surface normal;
- x' is the point on surface

Recursive Ray Tracing: assume only significant irradiance is in directions of light sources, specular reflection, and refraction;
$I = I_E + K_A I_{AL} + \sum_L (K_D(N \cdot L)I_L + K_S(V \cdot R)^n)I_L S_L + K_S I_R + K_T I_T$

- I_R is Radiance for mirror reflection ray
- I_T is radiance for refraction ray

Distributed Ray Tracing:estimate integral for each reflection by sampling incoming directions. $L_0(x', w') = L_e(x', w') + \sum_{samples} f_r(x, w, w')(w \cdot n)L_i(x', w)dw$

Monte Carlo Path Tracing: estimate integral for each pixel by sampling paths from camera

Radiosity: indirect diffuse illumination; light leaving all directions combined; *equations in cheatsheet*

- can discretize, by discretizing the surfaces into "elements"

3D Polygon Rendering: we can render polygons faster if we take advantage of *spatial coherence*; use a pipeline to draw 3D primitives into a 2D image

- **Modeling Transformation**: transform from 3D object coordinates into 3D world coordinate system
- Lighting: illuminate according to lighting and reflectance
- **Viewing Transformation**: transform into 3D camera coordinate system; origin -> eye, Z -> Back, Y -> Up, X -> Right
- **Project Transformation**: transform into 2D camera coordinate system

- Clipping: clip primitives outside camera view
- **Viewport Transformation**: transform 2D geometric primitives from screen coordinate system (normalized device coordinates) into image coordinate system (2D, pixels)
- scan conversion: draw pixels with texturing and hidden surface …
    - triangle sweep-line algorithm:
        - take advantage of spatial coherence; Compute which pixels are inside using horizontal spans! Process horizontal spans in scan-line order
        - take advantage of edge linearity, Use edge slopes to update coordinates incrementally

Rasterization: the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels or dots) for output on a video display or printer, or for storage in a bitmap file format. mostly in hardware.

- scan conversionL determine which pixels to fill
- shaing: determine a color for each filled pixel
    - ray casting: Simplest shading approach is to performindependent lighting calculation for every pixel;
    - polygon shading: Can take advantage of spatial coherence! Illumination calculations for pixels covered by sameprimitive are related to each other
    - **flat shading**: one illumination calculation per polygon; Assign all pixels inside each polygon the same color; objects look like they are composed of polygons
    - **Gouraud**: bilinear interpolation of colors at vertices, one lighting calculation per vertex, using barycentric coords; smooth shading over adjacent polygons, curved surfaces, soft shadows; piecewise linear approximation, need fine mesh to capture subtle lighting effects;
    - **Phong shading**: one lighting calculation per pixel, approximate surface normals for points inside polygons by bilinear interpolation of normals from vertices;

- texture mapping: describe shading variation within polygon interiors; vary pixel colors according to values fetched from a texture image; add visual detail to surfaces of 3D objects; 2D projective transformation, texture coordinate system to image coordinate system
- visible surface determination: figure out which surface is front-most at every pixel
  - "Painter's algorithm" - depth sort, $O(n \log n)$
    - sort surfaces in order of decreasing maximum depth
    - scan convert surfaces in back-to-front order, overwriting pixels
  - **Z-Buffer**: maintain color & depth of closest object per pixel, update only pixels with depth closer than in z-buffer; depths are interpolated form vertices, just like colors

Texture Filtering:

- must sample texture to determine color at each pixel in image
- might cause aliasing -> use elliptically shaped convolution filters ideally; in practice, use rectangles or squares;

Mipmap: keep textures pre-filtered at multiple resolutions; usually powers of 2; for each pixel, linearly interpolate between two closest levels (i.e., trilinear filtering); fast. easy for hardware;

Summed-area tables: at each texel keep sum of all values down and right; midmaps are more common

Bump Mapping: use gradient of grayscale image

Normal mapping: encode normals in RGB

Non-photorealistic rendering: provides control over style and abstraction;

Describe Shape-Conveying Lines:

- image-space features: intuitive motivation; well-suited for GPU; but difficult to stylize
  - edges: local maxima of gradient magnitude, in gradient direction
  - ridges/valleys: local minima/maxima of intensity, in direction of max Hessian

        eigenvector

- object-space features
    - view-independent: intrinsic properties of shape; can be pre-computed; but under changing view, can be misinterpreted as surface markings
        - topo lines: constant altitude form lines
        - creases: infinitely sharp folds
        - ridges and valleys (crest lines): local maxima of curvature
    - View-dependent object-space lines: seem to be perceived as conveying shape; but must be recomputed per frame

Silhouettes: boundaries between object and background

occluding contours: locations of depth depth discontinuities (aka interior and exterior silhouettes); surface normal perpendicular to view direction for smooth shapes

"almost contours": points that become contours in nearby views

Disocclusions: changes in visibility; a region which has been covered/obscured from view in the previous image has now become visible in the current image

- partial solution: use more photographs; OR, fill holes by interpolating nearby pixels
- Better solutions: multiple samples per pixel at different depths;

Image-based Rendering:

- Advantages: Photorealistic - by definition; Do not have to create 3D detailed model; Do not have to do lighting simulation; Performance independent of scene
- Disadvantages: Static scenes only! Real-world scenes only; Difficult for scenes with specularities, etc.; Limited range of viewpoints; Limited resolution

Animation

animation (is to make objects change over time according to scripted actions); simulation/dynamics is to predict how objects change over time according to physical laws;

Keyframing: define character poses at specific time steps called "keyframes"; interpolate variables describing keyframes to determine poses for character in between:

- linear interpolation: usually not enough continuity
- spline interpolation: maybe good enough

Articulated Figures: character poses described by set of rigid bodies connected by "joints"; good for humanoid characters; focus on joint angles.

forward kinematics describes motion of articulated character; animator specifies joint angles and computer finds positions of end-effector X. Joint motions are specified by perhaps spline curves

Inverse kinematics: happens when animators know the end-effector positions; so animator specifies end-effector positions X (possibly specified by spline curves), and computer finds joint angles.

- problems: system of equations is usually under-constrained and you can have multiple solutions

- Solution: find best solution (e.g. by minimizing energy in motion), or use non-linear optimization

Kinematics: Animator specifies poses (joint angles or positions)at keyframes and computer determines motion by kinematics and interpolation

- advantage: simple to implement; complete animator control
- Disadvantage: motions may not follow physical laws; tedious for animator

Dynamics: simulation of physics ensure realism of motion; animator specifies constraints, and computer finds the "best" physical motion satisfying constraints

- Advantages: Free animator from having to specify details of physically realistic motion with spline curves; Easy to vary motions due to new parameters and/or new constraints
- Challenges: Specifying constraints and objective functions; Avoiding local minima during optimization

Motion capture: measure motion of real characters and then simply "play it back" with kinematics, provides tools for animator to edit it

- Advantages: physical realism
- Challenges: animator control

Skinning:

Rigging: assigning weights; smoothness of skinned surface depends on smoothness of weights

Dynamics types:

- passive: no muscles or motors
- active: internal source of energy

Particle Systems: use many particles to model complex phenomena: keep array of particles and obey Newton's laws

Forward **Euler** Integration: update position and velocity at each step; but accuracy decreases as $\triangle t$ gets bigger; we can use **Midpoint method (2nd order Runge-Kutta)** to compute Euler step, evaluate f at the midpoint of Euler step, and then compute new position/velocity using midpoint velocity/acceleration; or we could try using adaptive step size

Particle System Forces *add in cheatsheet*