Computer Graphics I

Lecture 17: Review

Xiaopei LIU

School of Information Science and Technology
ShanghaiTech University

What is computer graphics?

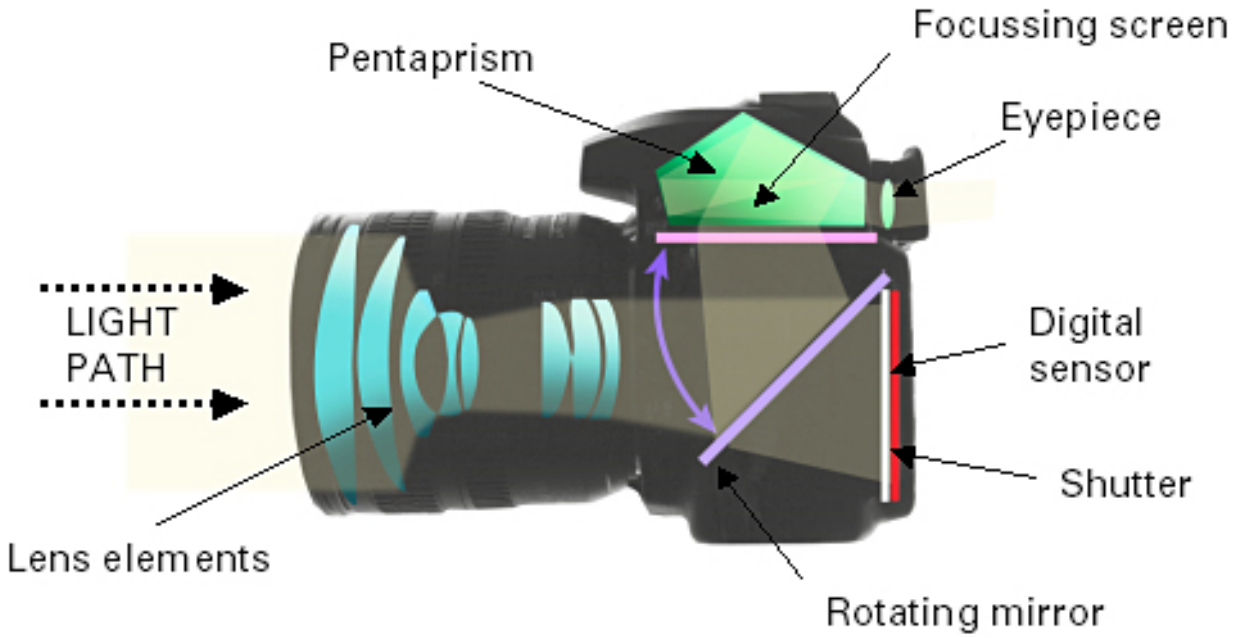
- **Common definition**
  - Pictures and films created using computers
- **Broader definition**
  - Almost everything on computers that are not text or sound

What is a (digital) image?

- **Image**
  - An artifact that depicts visual perception
- **Digital image**
  - A numeric representation of a (2D) image (rasterization)
  - Raster images have a finite set of picture elements or pixels

How to obtain a digital image?

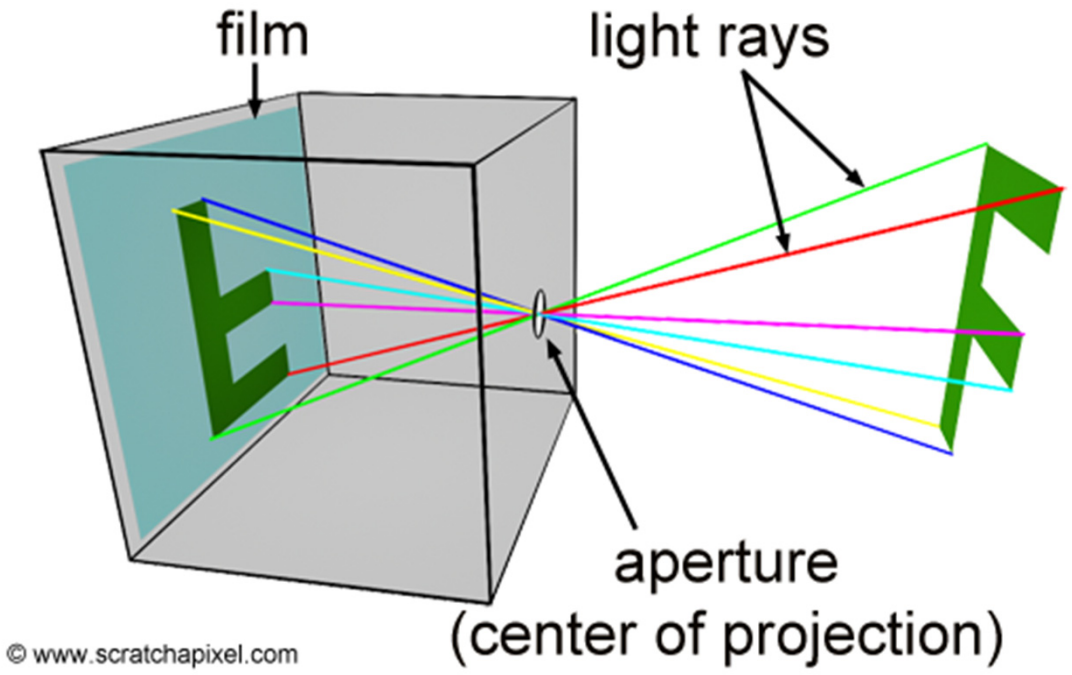
- **Imaging device (camera)**



1. Camera model

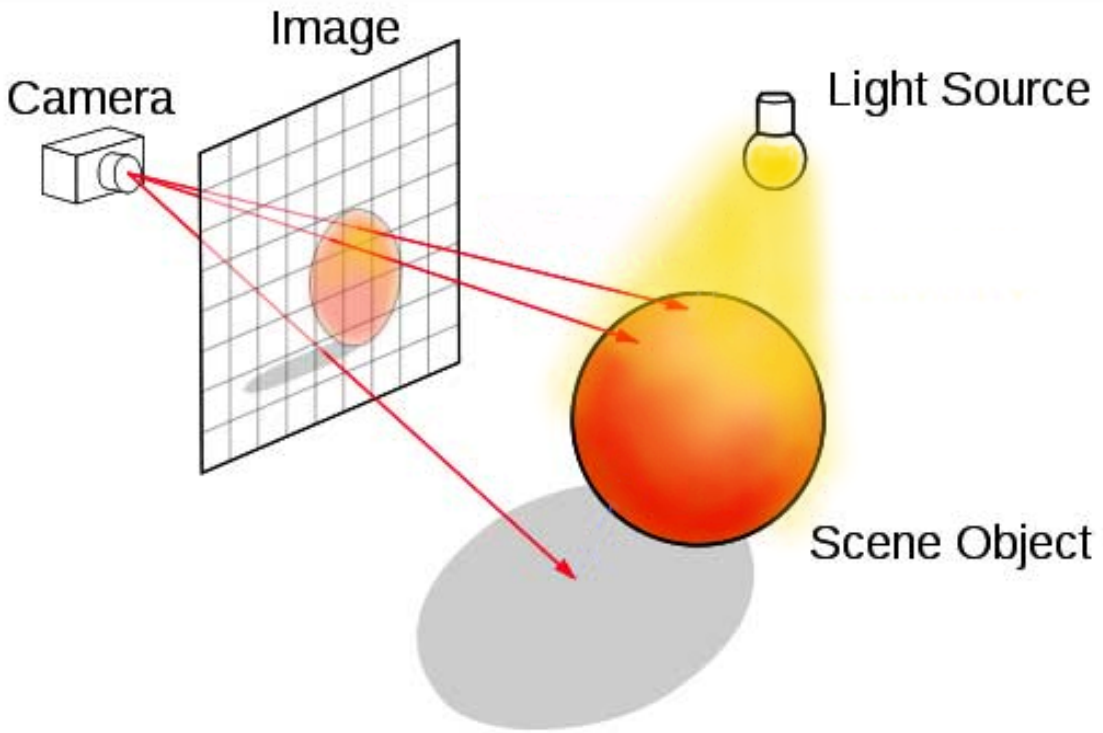
Camera model

- **Pin-hole camera**



Camera model

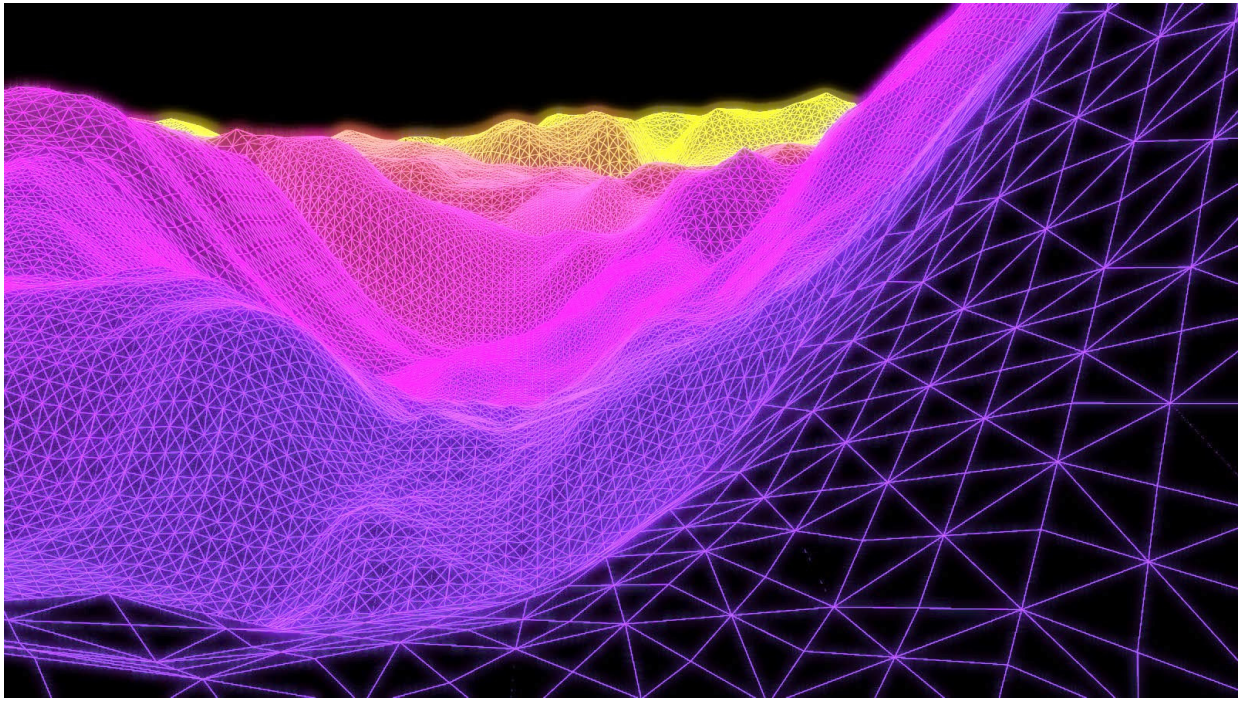
- **Pin-hole camera model**



2. Geometric representation & modeling

Mesh

- **Representation of shapes with a collection of geometrical primitives**
  - Triangles/quadrilaterals
  - Vertices, faces, normals, collectivity (topology)



Mesh organization

- **Face-vertex representation**

Face-Vertex Meshes

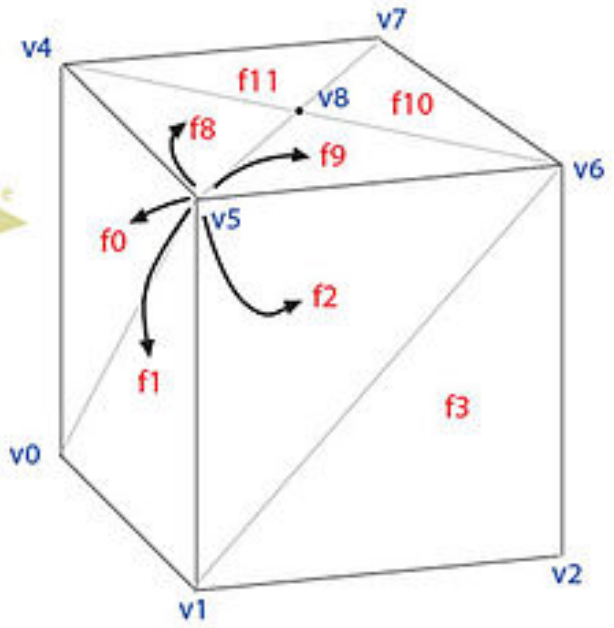
Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vertex List

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15

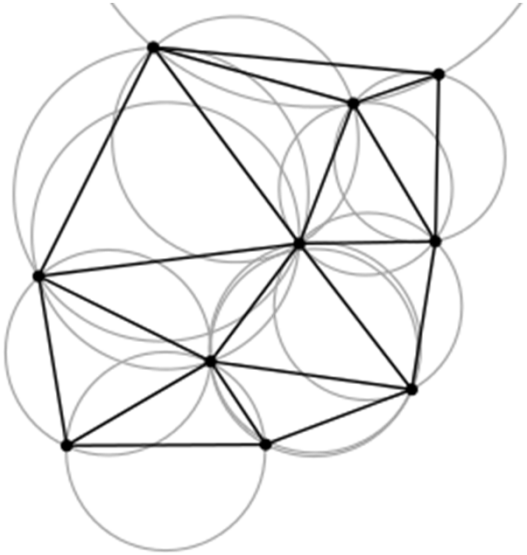
example



Mesh generation -Delaunay triangulation

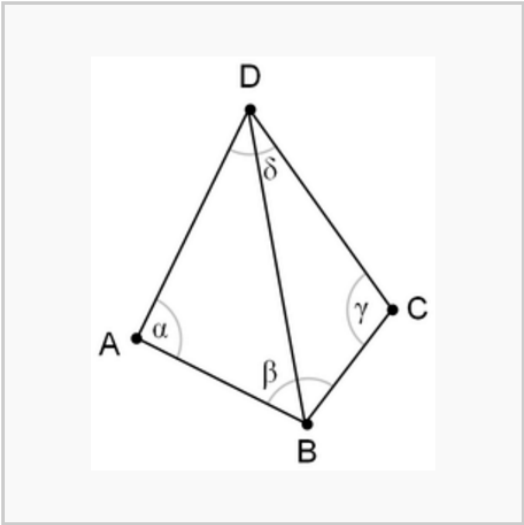
- **Delaunay triangulation for a point-set P:**
  - A triangulation DT(**P**) such that no point in **P** is inside the circumcircle of any triangle in DT(**P**)
  - Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation

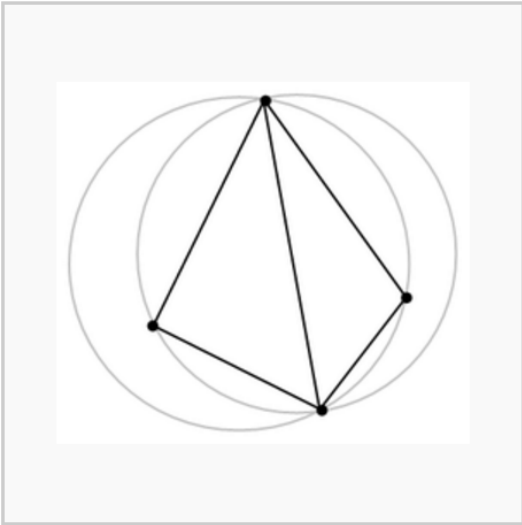
每个三角形的外接圆不包含其他的顶点
使最小角最大

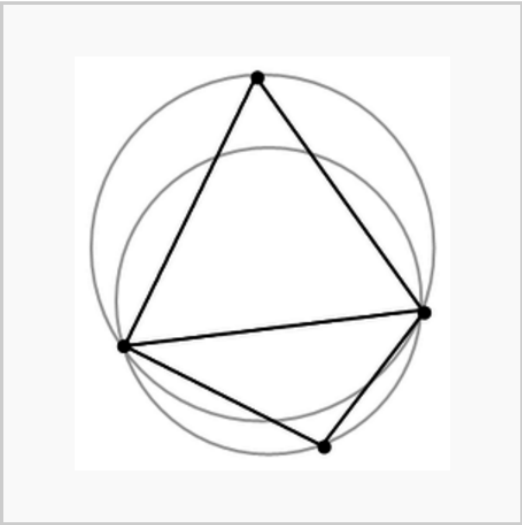


Delaunay triangulation - Flipping

- **Flipping**
  - If two triangles do not meet the Delaunay condition, switching the common produces two triangles with Delaunay condition



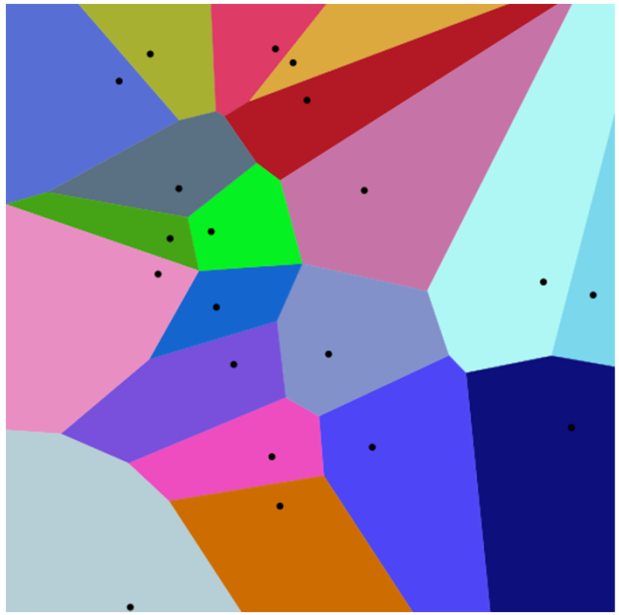




Delaunay triangulation - Voronoi diagram

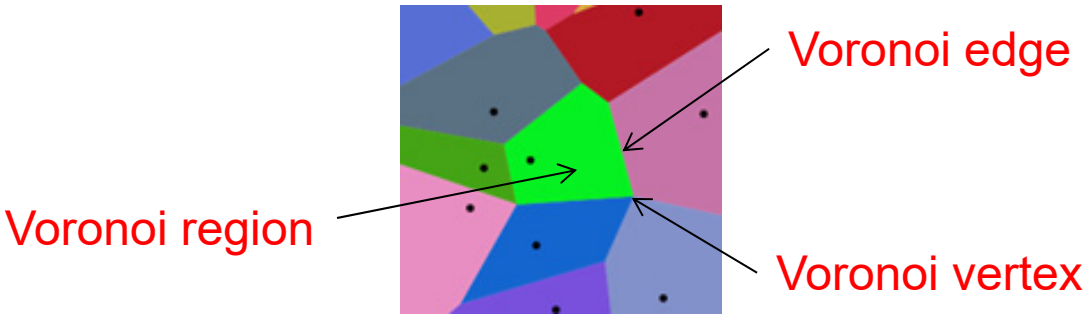
- **A Voronoi diagram is**
  - a partitioning of a plane into regions based on distance to points
  - for each seed there is a corresponding region consisting of all points closer to that seed than to any other

- **These regions are called *Voronoi cells***



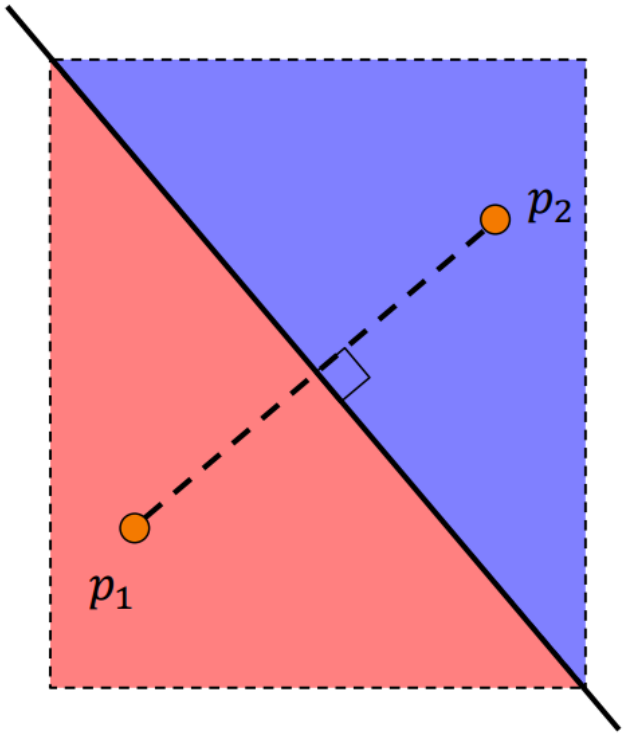
Delaunay triangulation - Voronoi diagram

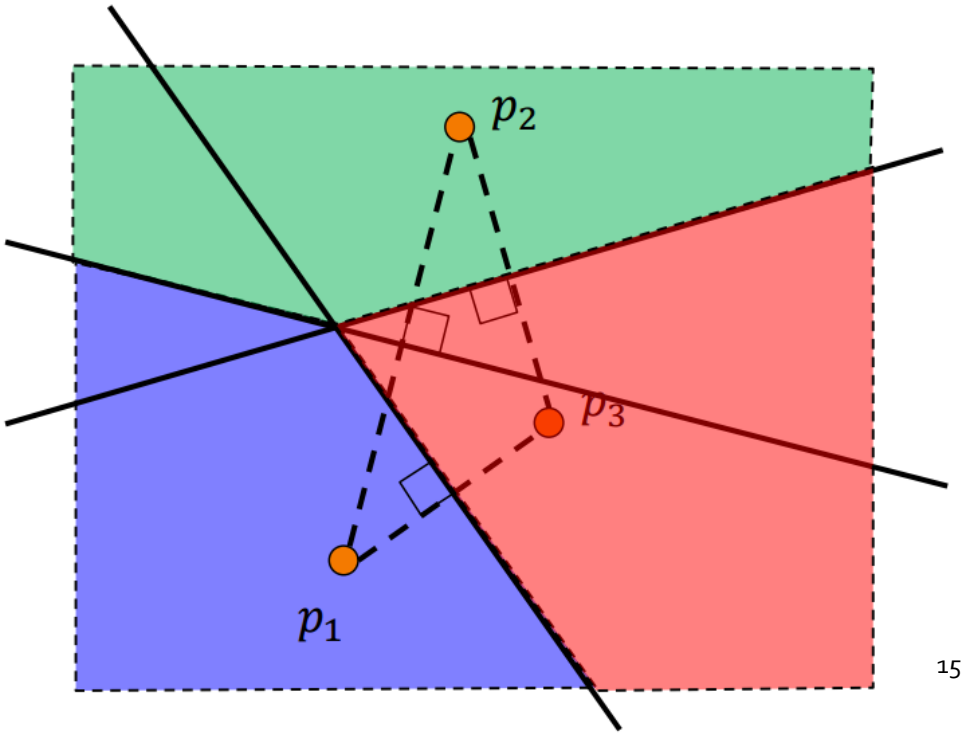
- **Voronoi region**
  - R(**S**; $\mathbf{s}_i$) is called the Voronoi region of $\mathbf{s}_i$
- **Voronoi edge**
  - The line segments shared by the boundaries of two Voronoi regions are called Voronoi edges
- **Voronoi vertex**
  - The points shared by the boundaries of three or more Voronoi regions



Delaunay triangulation - Voronoi diagram

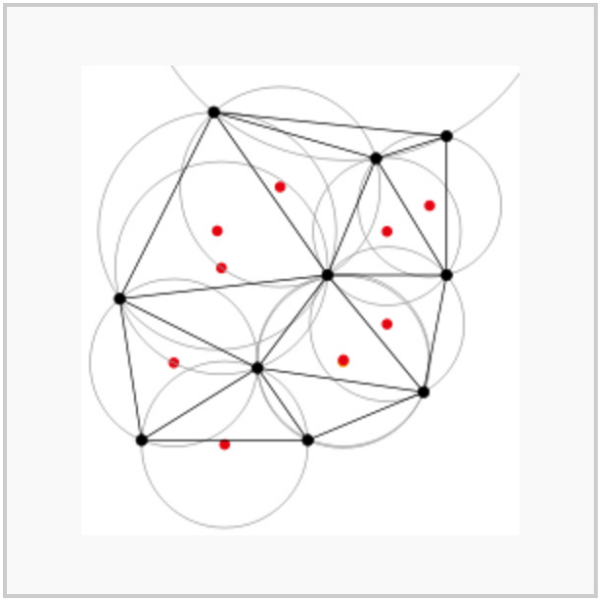
- **How to compute Voronoi Diagram?**
  - Half-space partition
  - For example, 2 points and 3 points partition

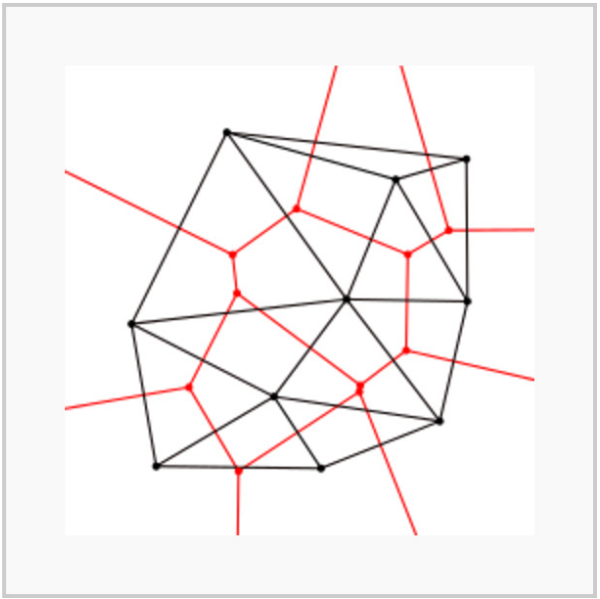




Delaunay triangulation - Voronoi diagram

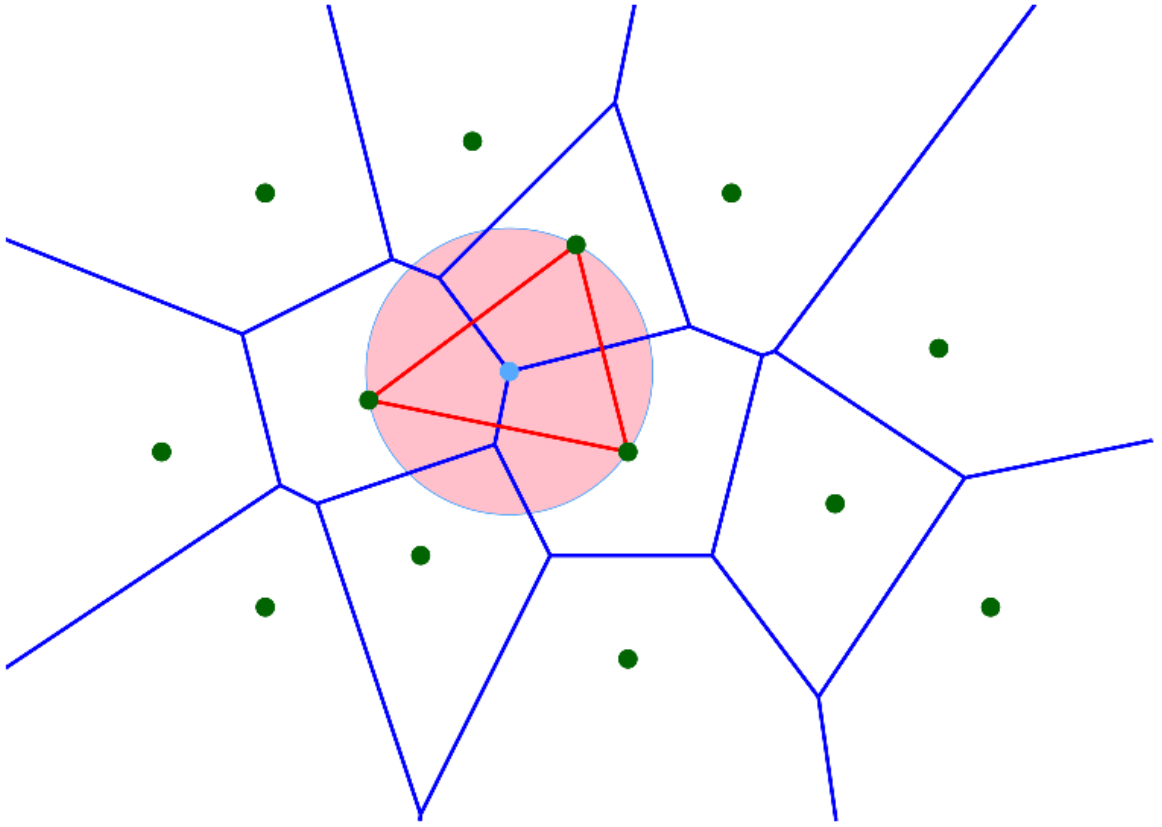
- **Why we need Voronoi diagram?**
  - Voronoi diagram of a set of points is dual to its Delaunay triangulation
  - Connecting the centers of the circumcircles produces the Voronoi diagram





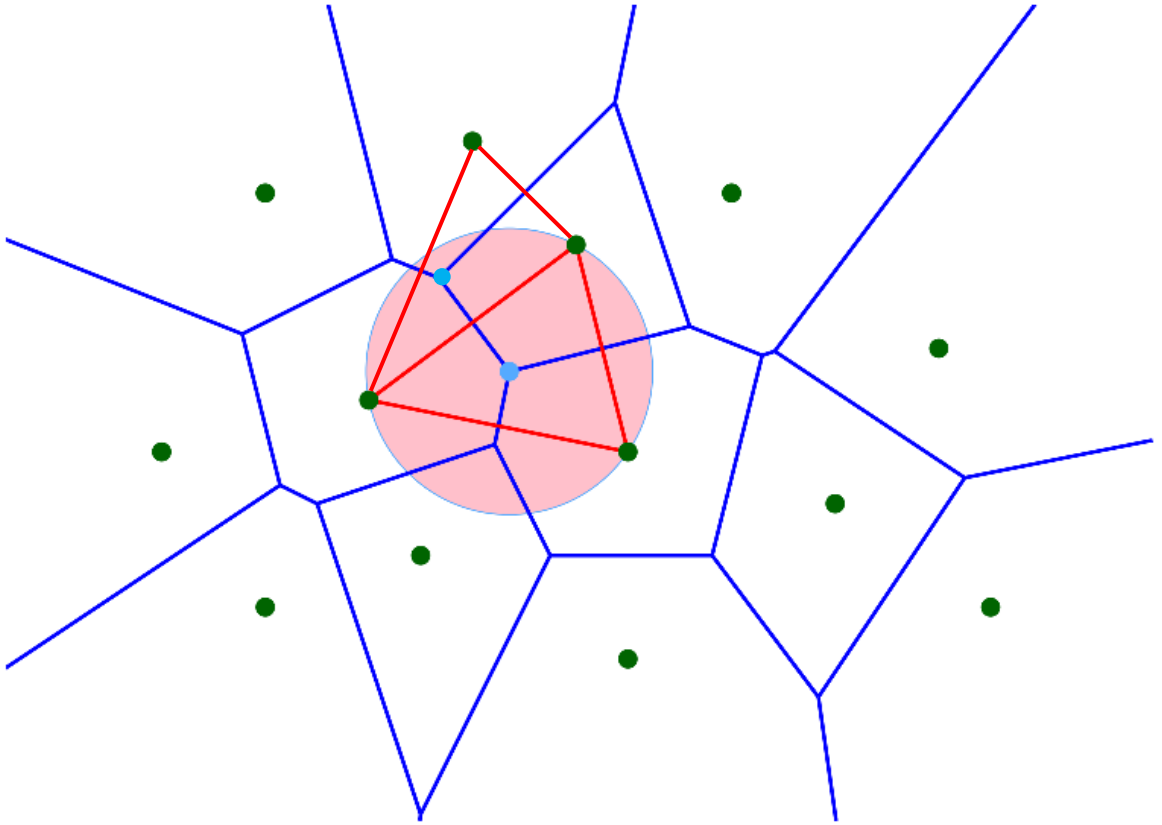
Voronoi diagram

- **From Voronoi diagram to Delaunay triangulation**



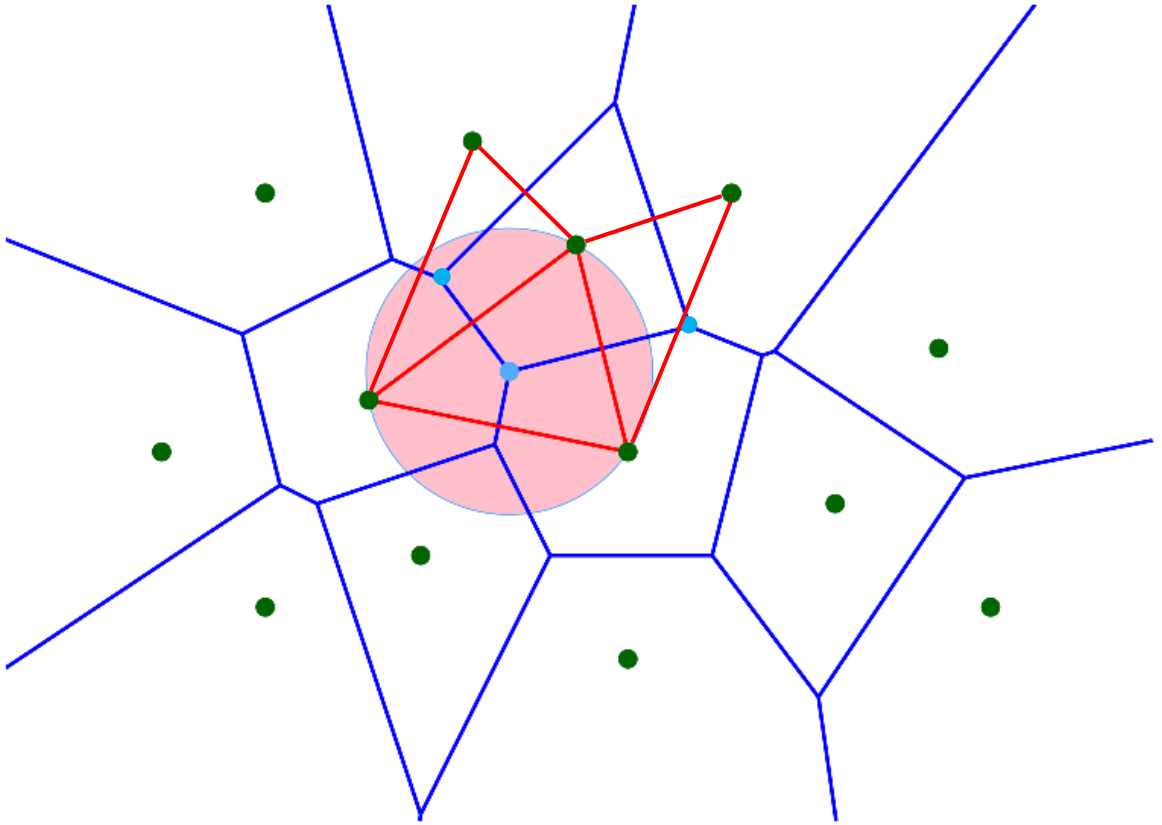
Voronoi diagram

- **From Voronoi diagram to Delaunay triangulation**



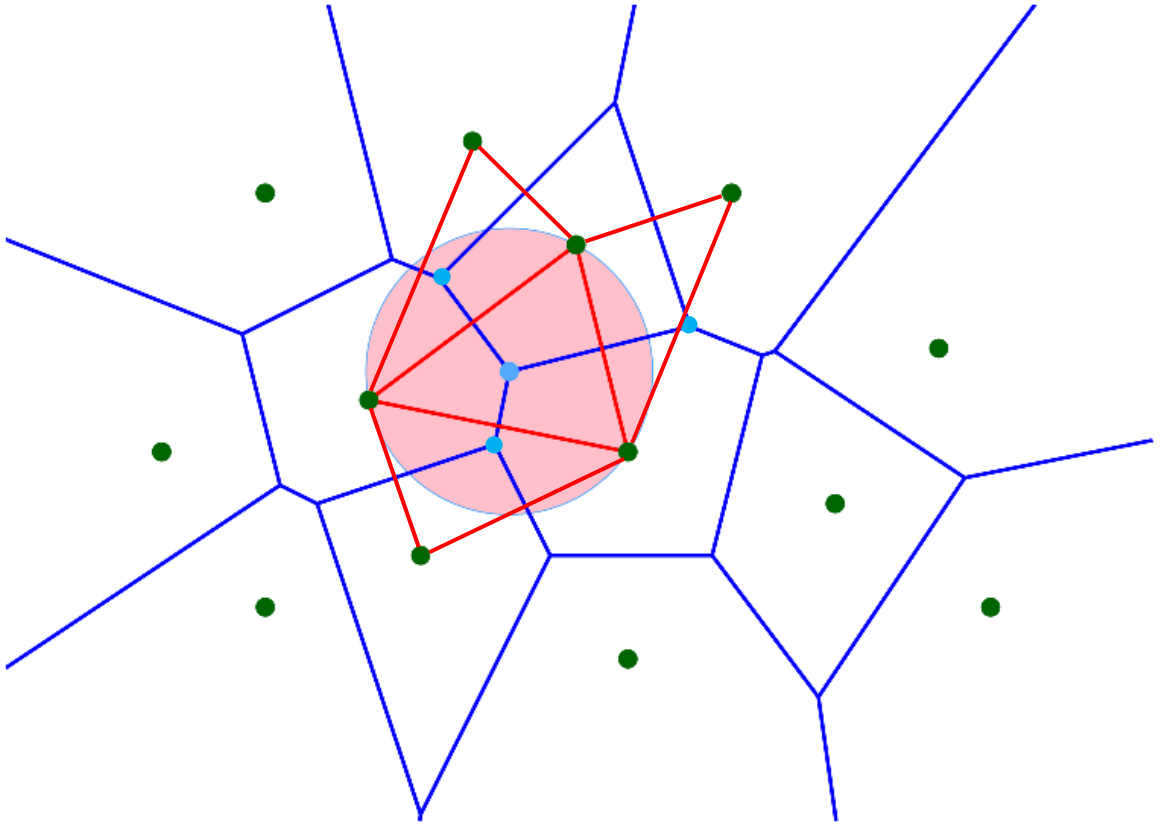
Voronoi diagram

- **From Voronoi diagram to Delaunay triangulation**



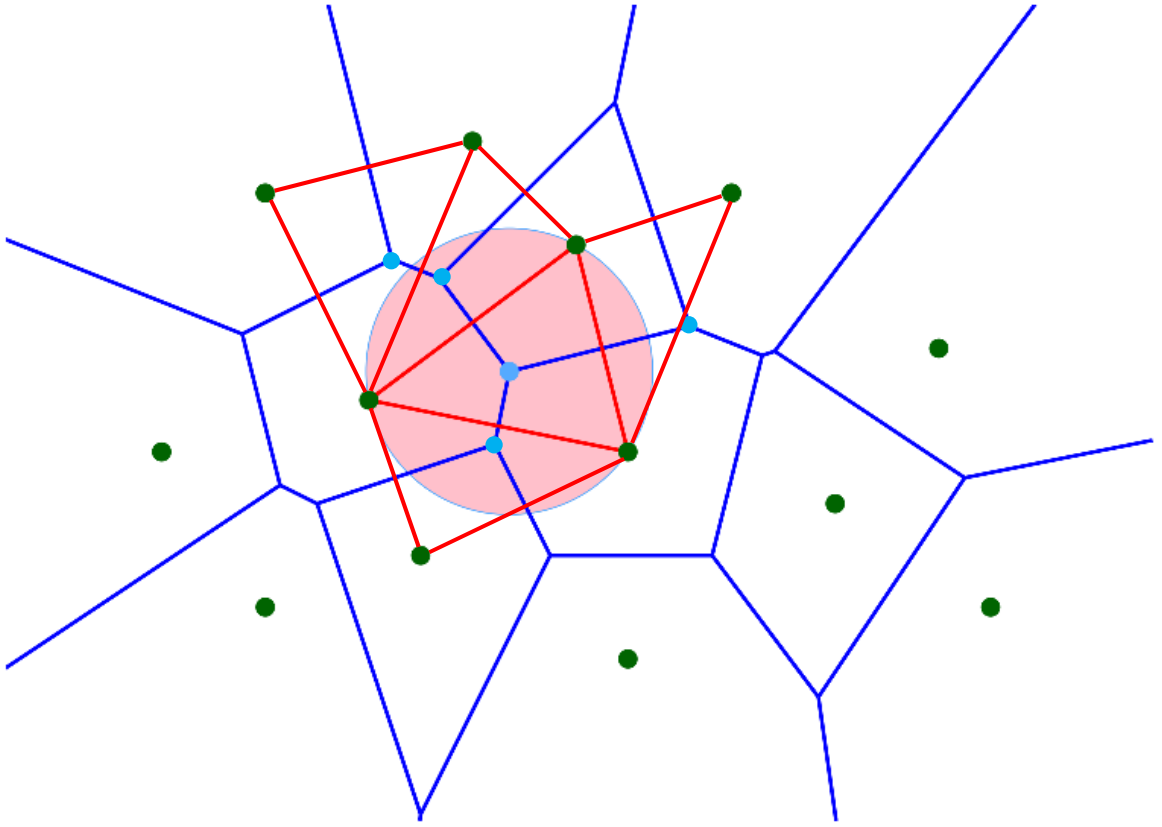
Voronoi diagram

- **From Voronoi diagram to Delaunay triangulation**



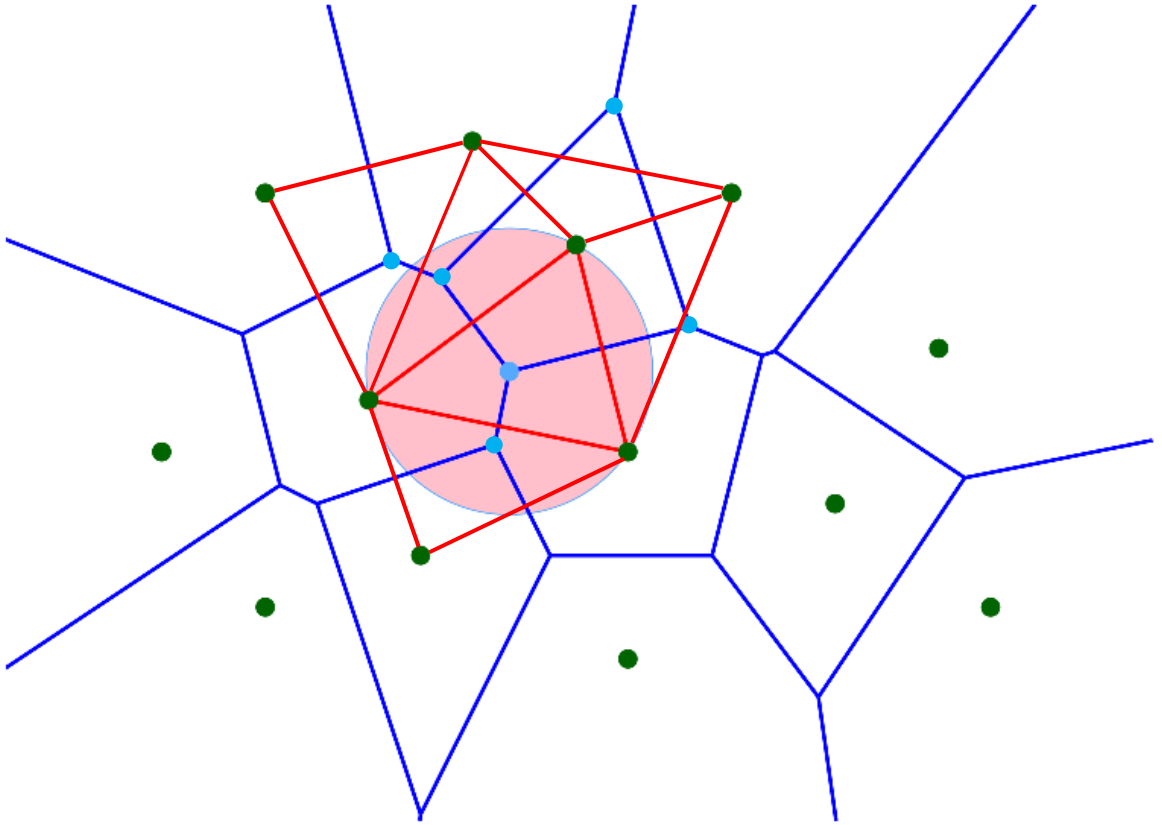
Voronoi diagram

- **From Voronoi diagram to Delaunay triangulation**



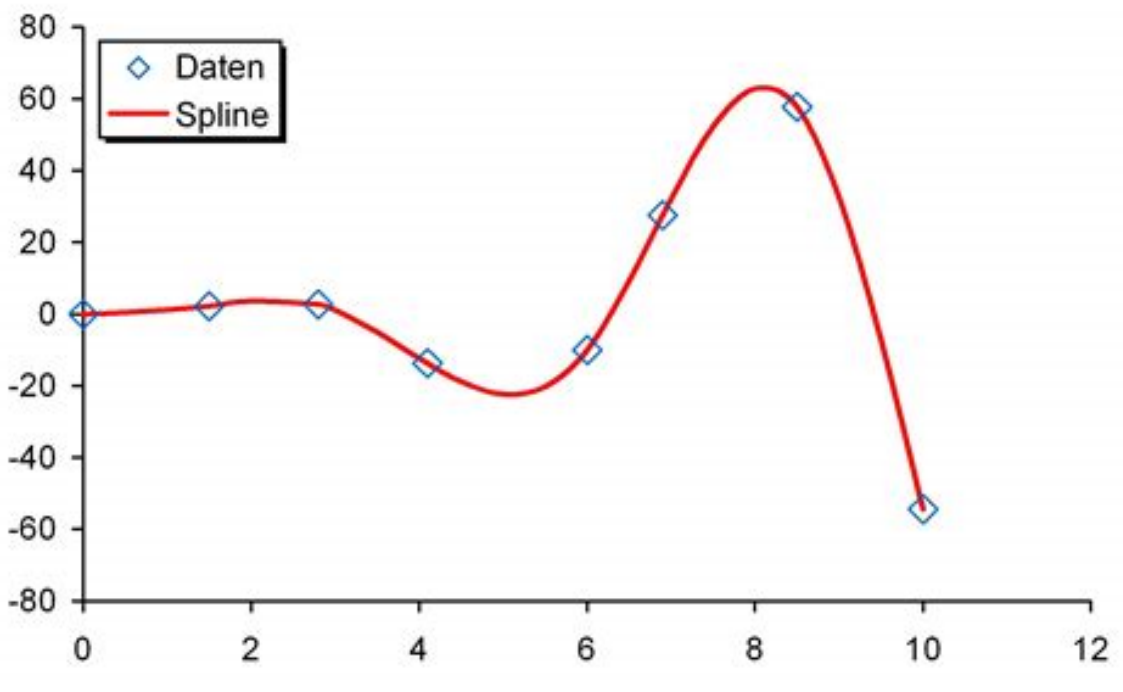
Voronoi diagram

- **From Voronoi diagram to Delaunay triangulation**



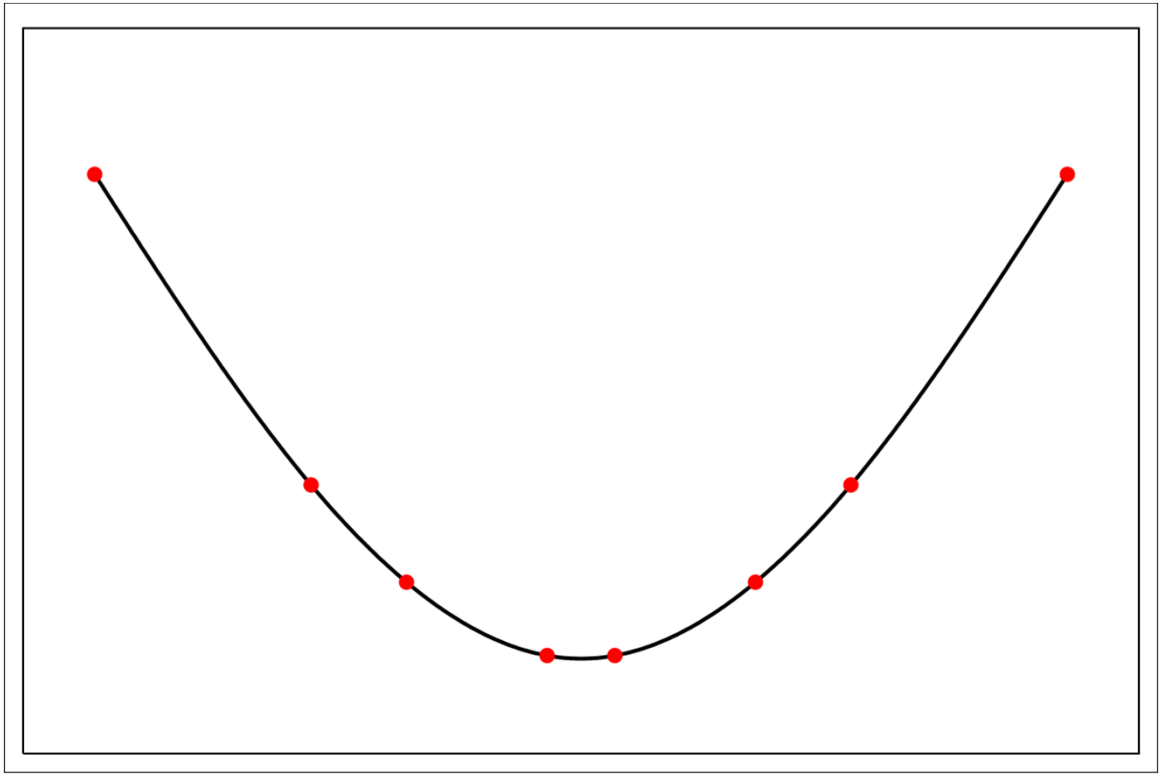
Spline

- **A spline is a special function defined piecewise by polynomials of low degree**
  - avoid Runge's phenomenon for more sample points
  - originally, high-degree polynomials should be used



Spline

- **Piecewise cubic spline**
  - The interpolation function is piecewise defined by cubic polynomials



Cubic spline interpolation

- **Given a function f (x) defined on [a, b] and a set of nodes**

$$a = x_0 < x_1 < x_2 < \cdots < x_n = b,$$

- **A cubic spline interpolant, S, for f is a piecewise cubic polynomial, S_j on $[x_j ; x_{j+1}]$ for j = 0, 1, … , n -1**

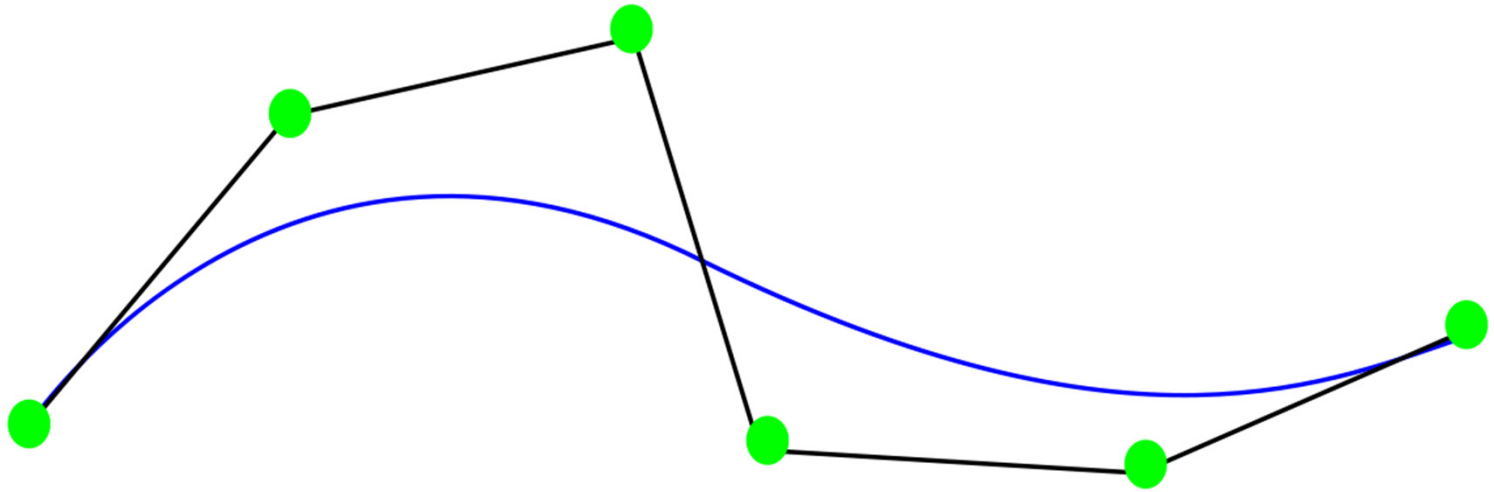
$$S(x) = \begin{cases} a_0 + b_0(x - x_0) + c_0(x - x_0)^2 + d_0(x - x_0)^3 & \text{if } x_0 \le x \le x_1 \\ a_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3 & \text{if } x_1 \le x \le x_2 \\ \vdots & \vdots \\ a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 & \text{if } x_i \le x \le x_{i+1} \\ \vdots & \vdots \\ a_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + d_{n-1}(x - x_{n-1})^3 & \text{if } x_{n-1} \le x \le x_n \end{cases}$$

Cubic spline interpolation

- **The cubic spline interpolant will have the following properties:**
  - $S(x_j) = f(x_j)$ for $j = 0, 1, \ldots, n$.
  - $S_j(x_{j+1}) = S_{j+1}(x_{j+1})$ for $j = 0, 1, \ldots, n-2$.
  - $S'_j(x_{j+1}) = S'_{j+1}(x_{j+1})$ for $j = 0, 1, \ldots, n-2$.
  - $S''_j(x_{j+1}) = S''_{j+1}(x_{j+1})$ for $j = 0, 1, \ldots, n-2$.
  - One of the following boundary conditions (BCs) is satisfied:
    - $S''(x_0) = S''(x_n) = 0$ (**free** or **natural** BCs).
    - $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$ (**clamped** BCs).

Polynomial approximation

- **The polynomials are generated by control points**
  - the curve does not necessarily pass through control points
  - control points are used to control the shape of the curve



Bézier curve

- **Linear Bézier curves**

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1,\ 0 \le t \le 1$$

- **Quadratic Bézier curves**

$$\mathbf{B}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2],\ 0 \le t \le 1$$

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2,\ 0 \le t \le 1$$

Bézier curve

- **Cubic Bézier curves**

$$\mathbf{B}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2}(t) + t\mathbf{B}_{\mathbf{P}_1,\mathbf{P}_2,\mathbf{P}_3}(t)$$

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3,\ 0 \le t \le 1$$

- **General definition**
  - recursive definition

$$\mathbf{B}_{\mathbf{P}_0}(t) = \mathbf{P}_0, \text{ and}$$

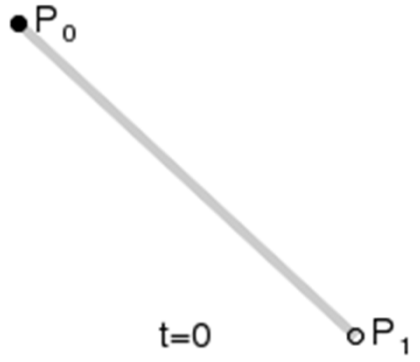
$$\mathbf{B}(t) = \mathbf{B}_{\mathbf{P}_0\mathbf{P}_1\ldots\mathbf{P}_n}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0\mathbf{P}_1\ldots\mathbf{P}_{n-1}}(t) + t\mathbf{B}_{\mathbf{P}_1\mathbf{P}_2\ldots\mathbf{P}_n}(t)$$

Bézier curve

- **Evaluation**
  - de Casteljau's algorithm
    - recurrence relation

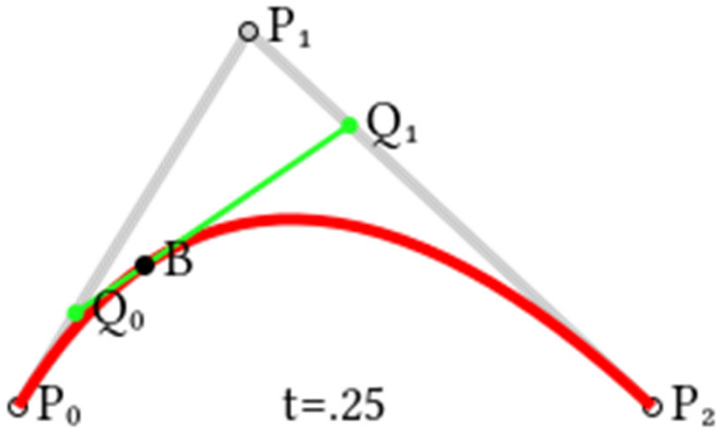
$$\beta_i^{(0)} := \beta_i, \ i = 0, \ldots, n$$
$$\beta_i^{(j)} := \beta_i^{(j-1)}(1 - t_0) + \beta_{i+1}^{(j-1)} t_0, \ i = 0, \ldots, n-j, \ j = 1, \ldots, n$$

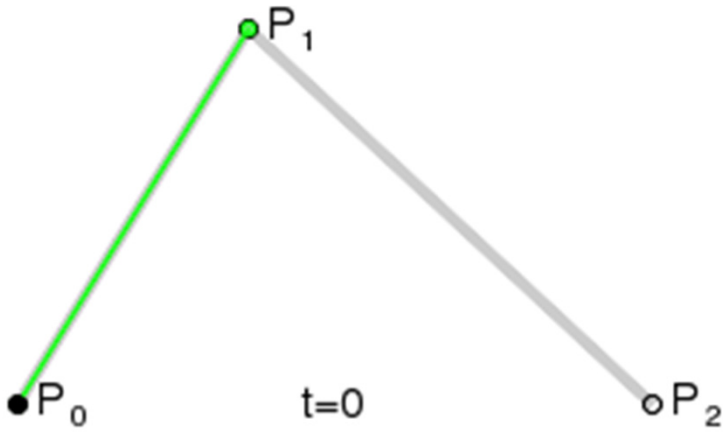
- **Linear curves**



Bézier curve

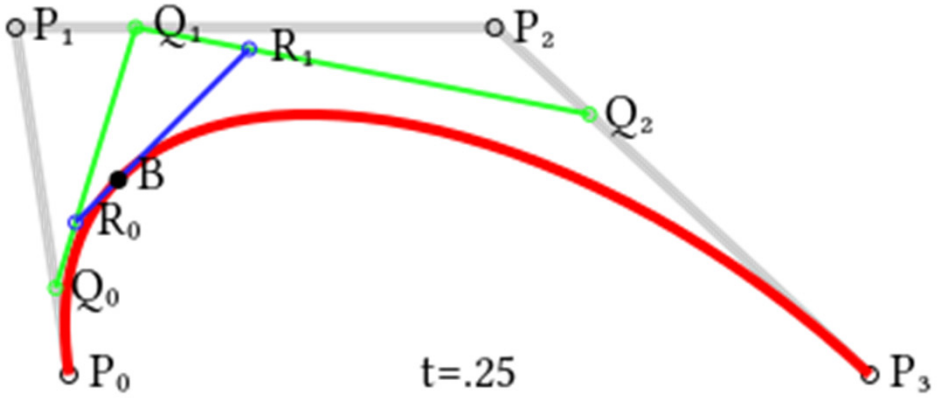
- **Quadratic curves**

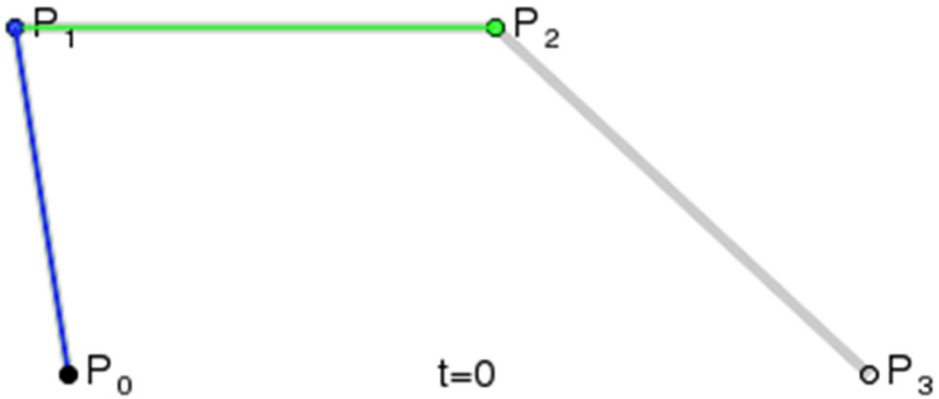




Bézier curve

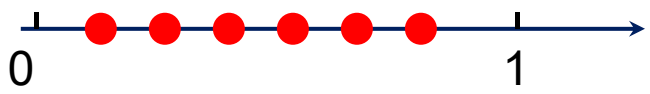
- **Higher-order curves**
  - cubic Bézier curve



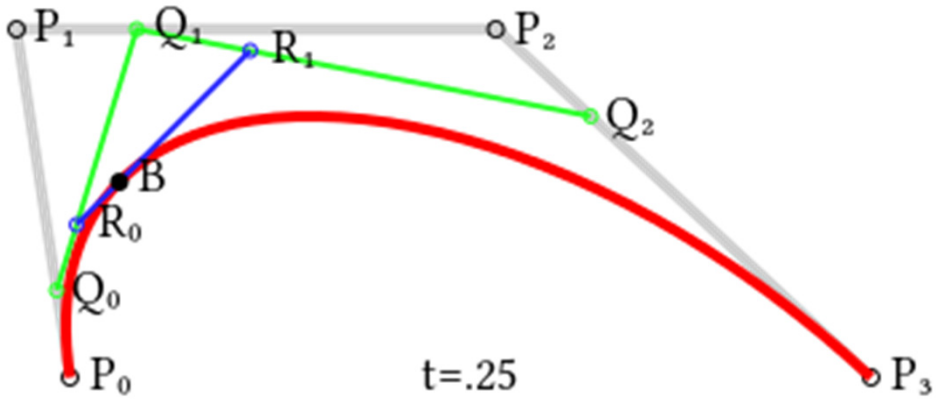


Meshing a Bézier curve

- **Meshing in parameter space**
  - Sample in parameter space and connect sample points by line segments

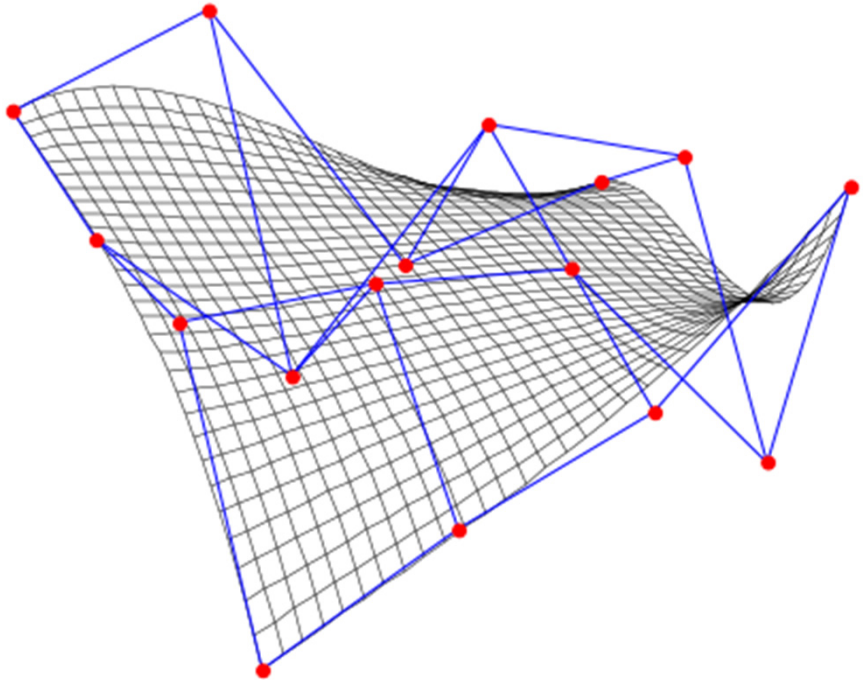


t parameter space



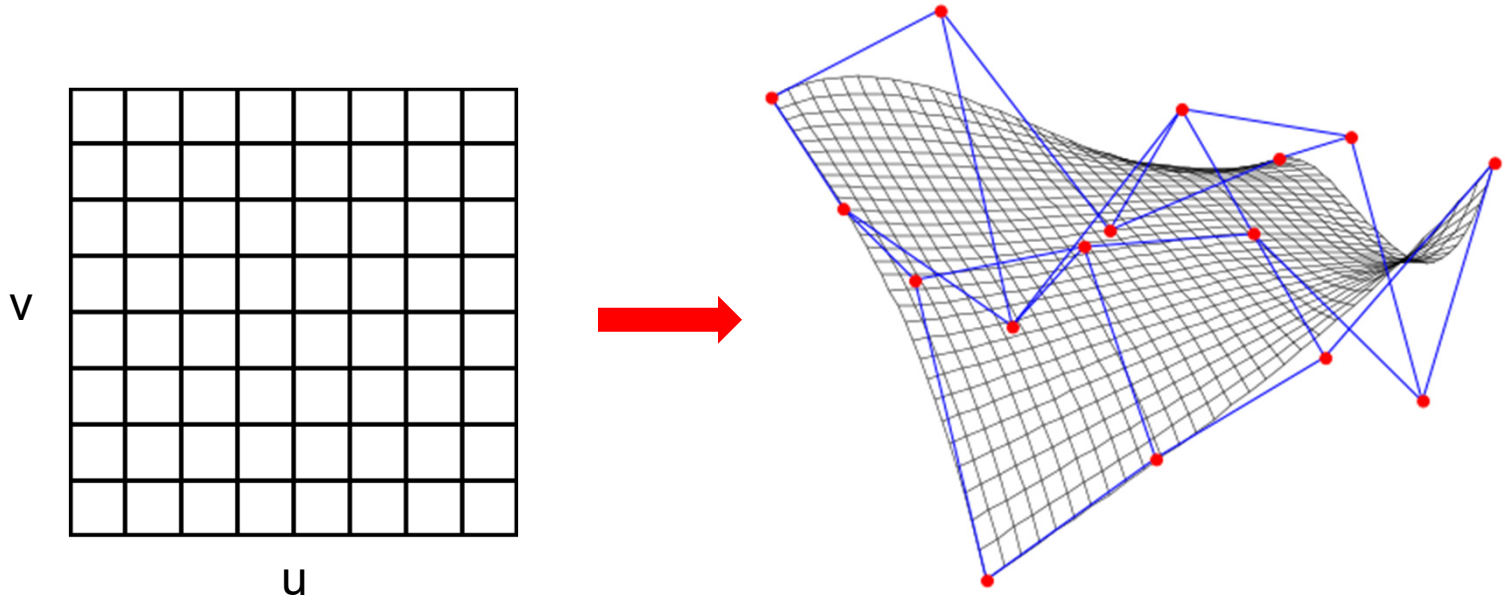
Bézier surface

- **A Bézier surface of degree (*n*, *m*) is defined by a set of (*n* + 1)(*m* + 1) control points $k_{i,j}$**
  - it maps the unit square into a smooth-continuous surface



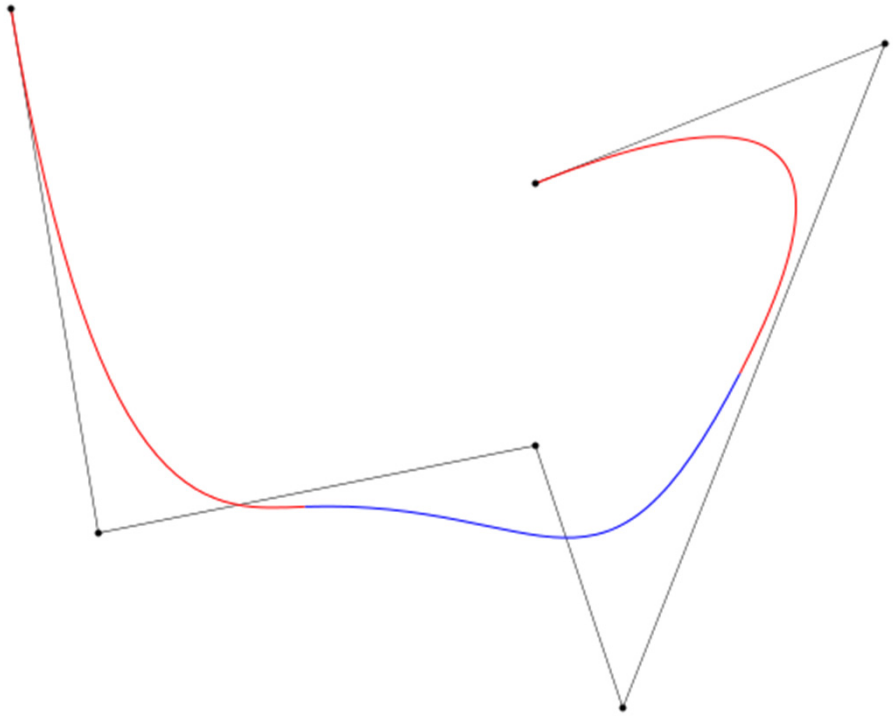
Meshing a Bézier surface

- **Meshing in parameter space**
  - gridding in u,v parameter space



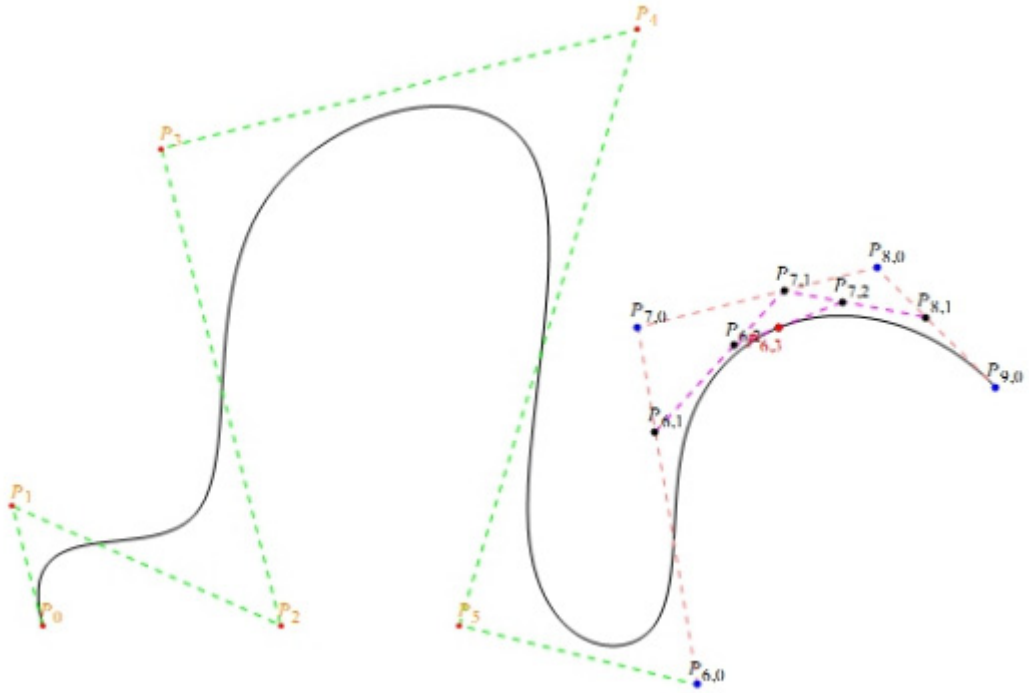
B-spline curve

- **Basis spline – B-spline**
  - a spline function that has minimal support with respect to a given degree



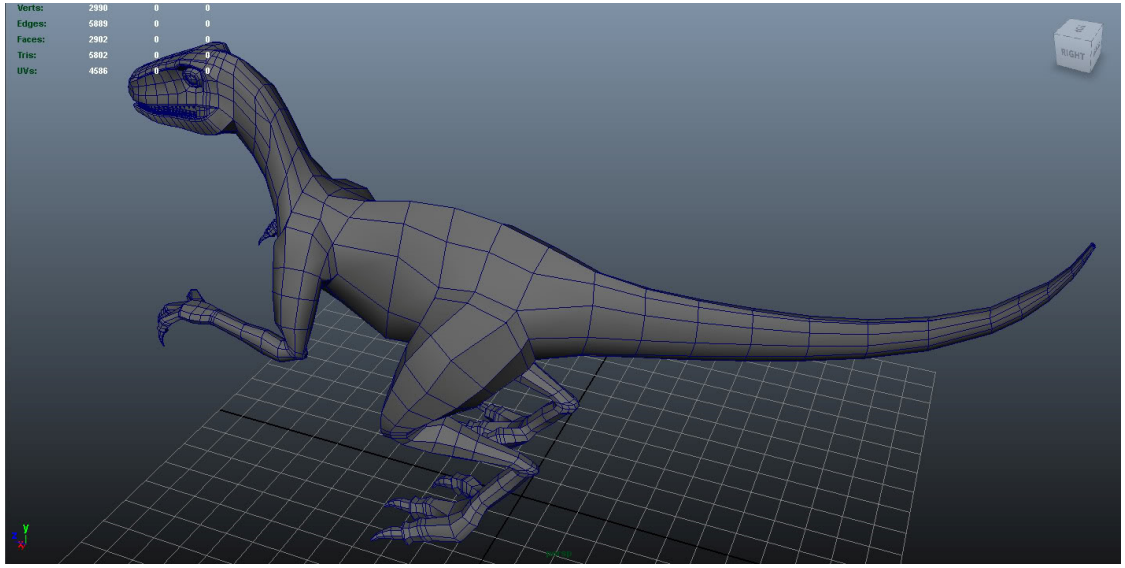
B-spline curve

- **Evaluation**
  - de Boor algorithm
  - find the support range of the current parameter
  - apply recursive evaluation like in Bézier curve evaluation



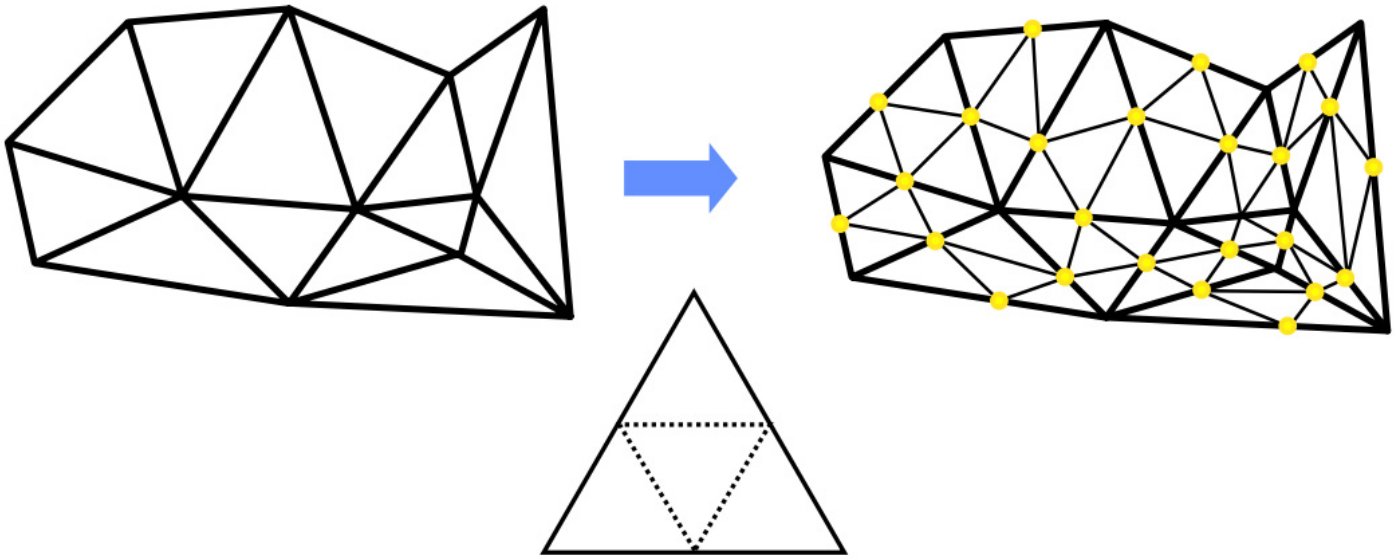
B-spline surface

- **Like Bézier surface, B-spline surface can be constructed with tensor-product**
  - meshing in u-v parameter space



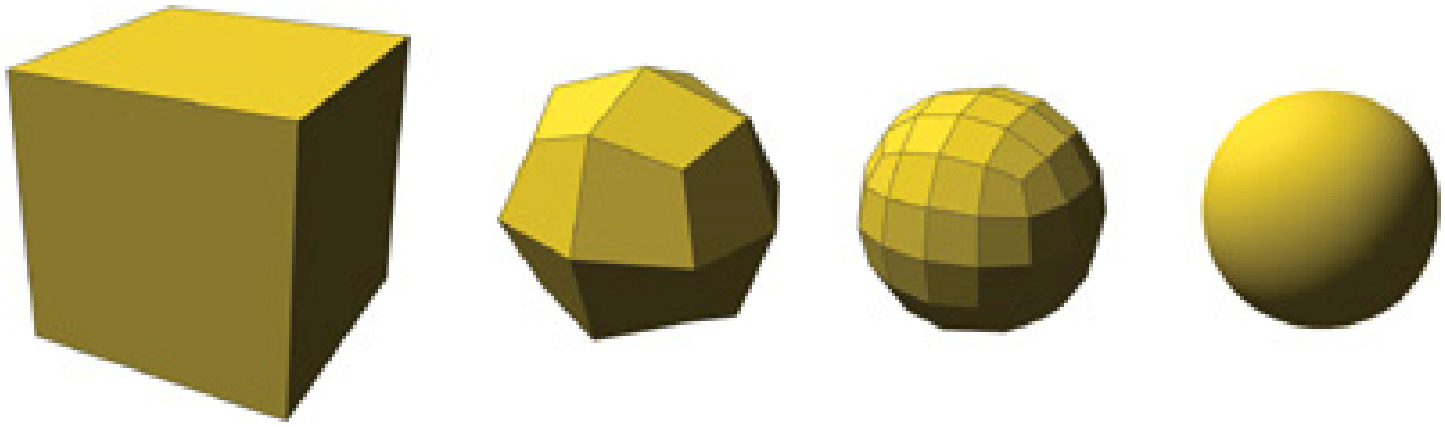
Mesh subdivision

- **Overview**
  - Subdivision surfaces are defined *recursively*
  - Starting with a given polygonal mesh, a (convergent) *subdivision scheme* is applied to this mesh



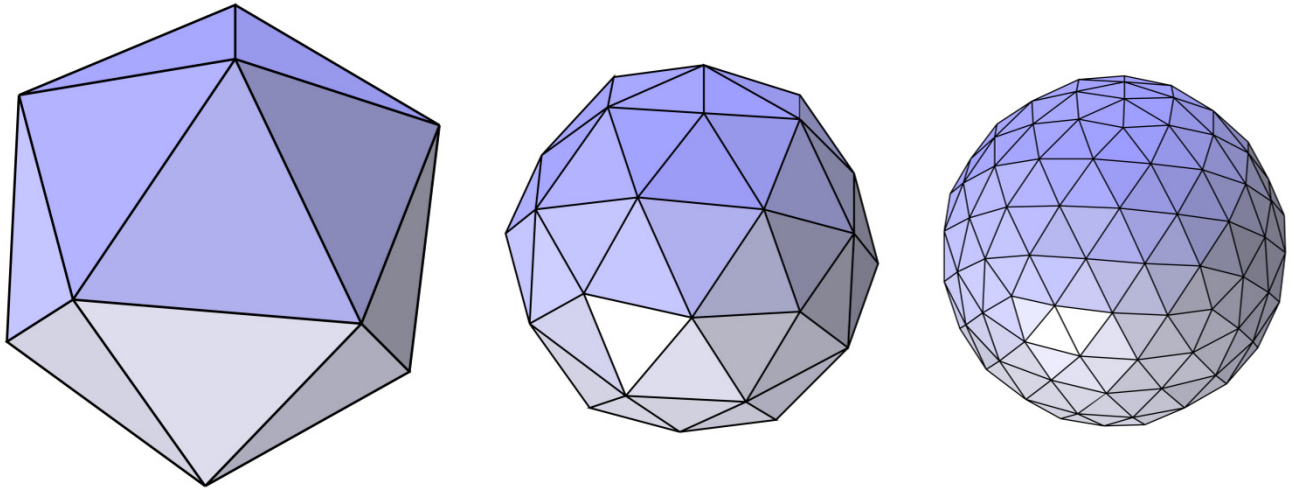
Mesh subdivision

- **Catmull–Clark subdivision scheme**
  - Devised by Edwin Catmull and Jim Clark in 1978
  - A generalization of bi-cubic uniform B-spline surfaces to arbitrary topology



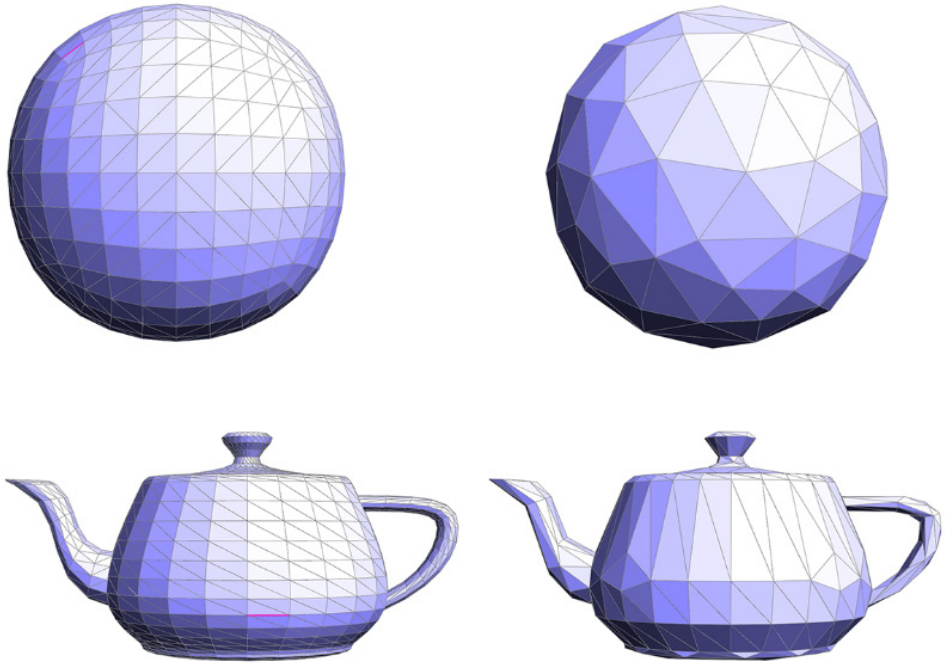
Mesh subdivision

- **Loop subdivision surfaces**
  - Quadrilateral based meshes generally use Catmull-Clark subdivision scheme
  - Triangle based meshes generally use loop subdivision



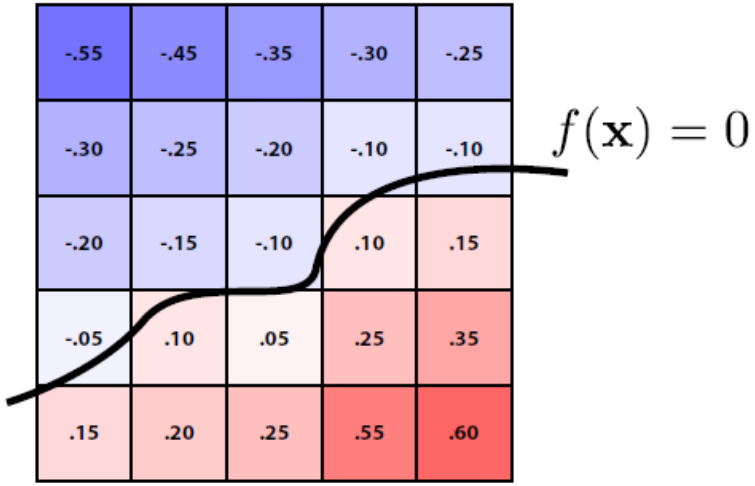
What is mesh simplification/coarsening?

- **The process to reduce the number of vertex/face of a polygonal mesh**
  - Approximate the same shape with fewer primitives
  - Inverse process of mesh subdivision/refinement



Implicit representation of a surface

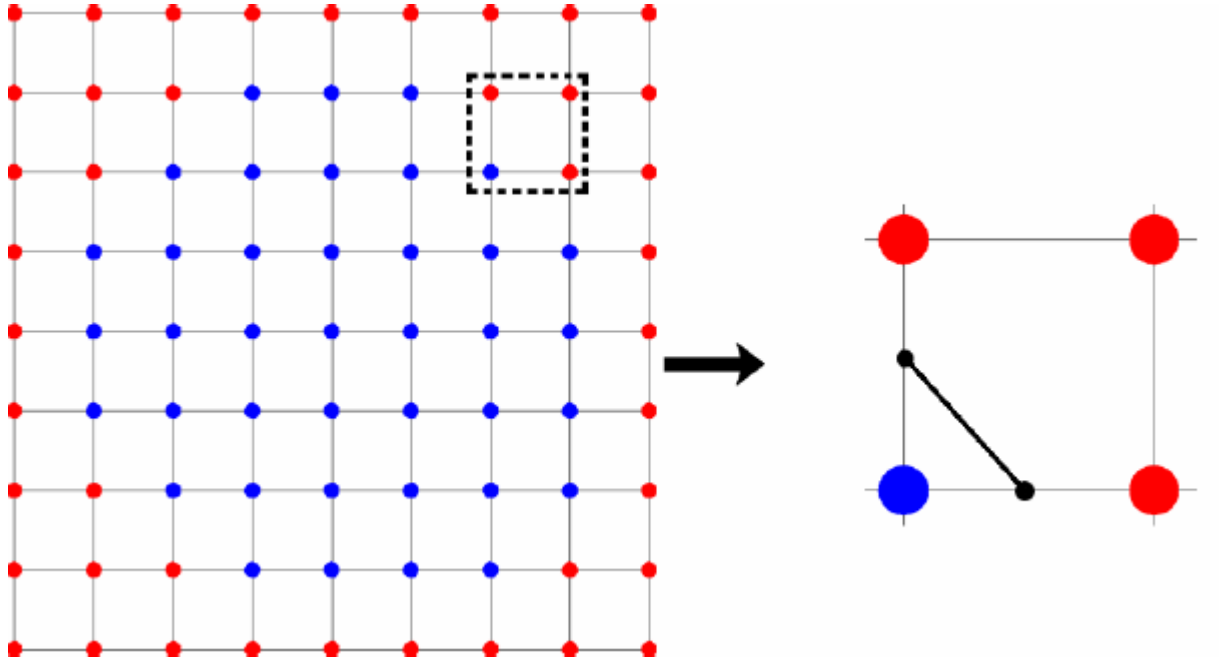
- Implicit surface representation
  - Implicit surfaces have some nice features (e.g., merging/splitting)
  - But, hard to describe complex shapes in closed form
  - Alternative: store a grid of values approximating function



  - Surface is found where *interpolated* values equal zero
  - Provides much more explicit control over shape (like a texture)

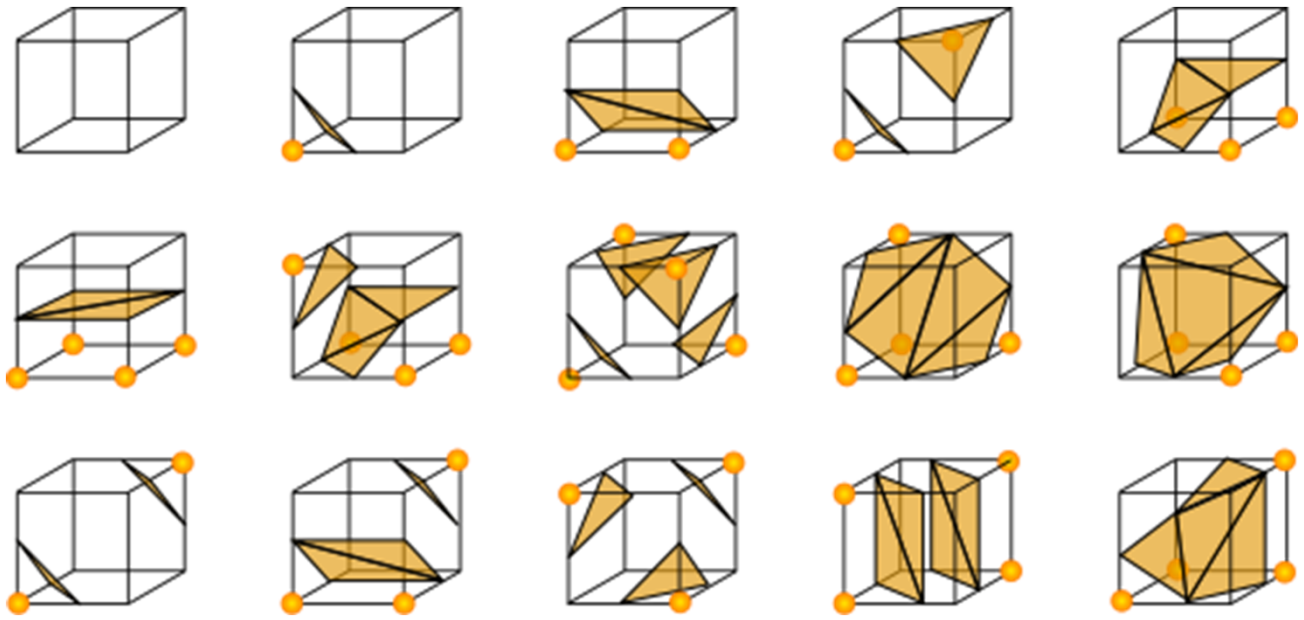
Construction of isosurface

- **Marching cubes**
  - 2D surface reconstruction from samples of a function
  - Then, we connect the two points to form an edge in isosurface



Construction of isosurface

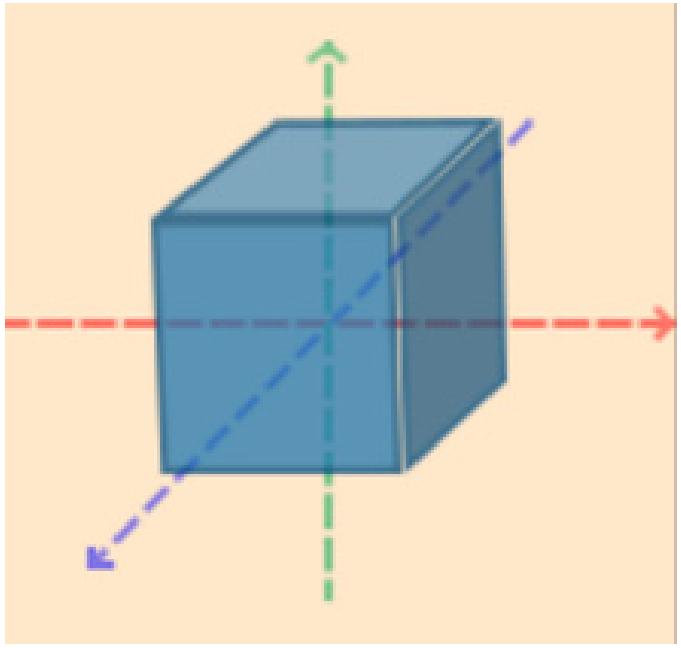
- **Marching cubes**
  - 3D surface reconstruction from samples of a function
  - All the constructed triangle faces in a cell form the whole isosurface



3. Geometric transformation

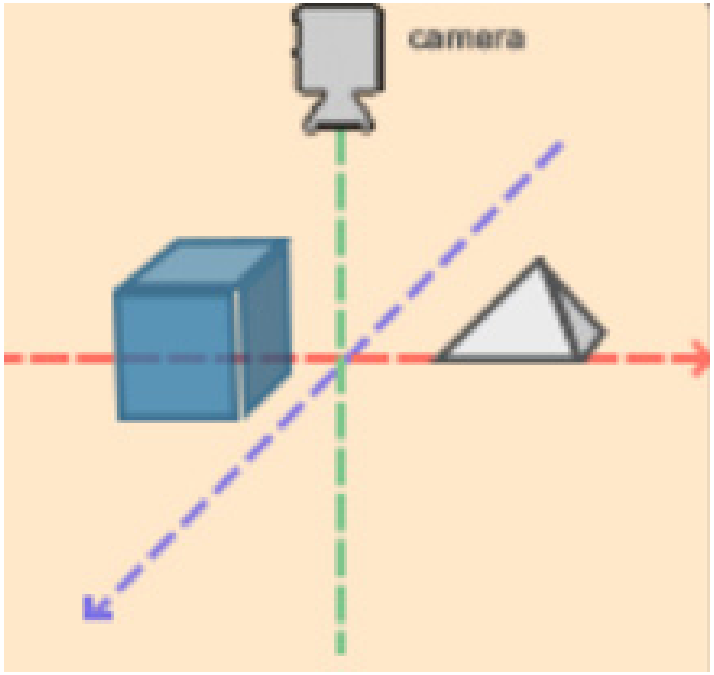
Coordinate spaces

- **Local(object) coordinate space**
  - Local coordinate space is the coordinate space that is local to your object



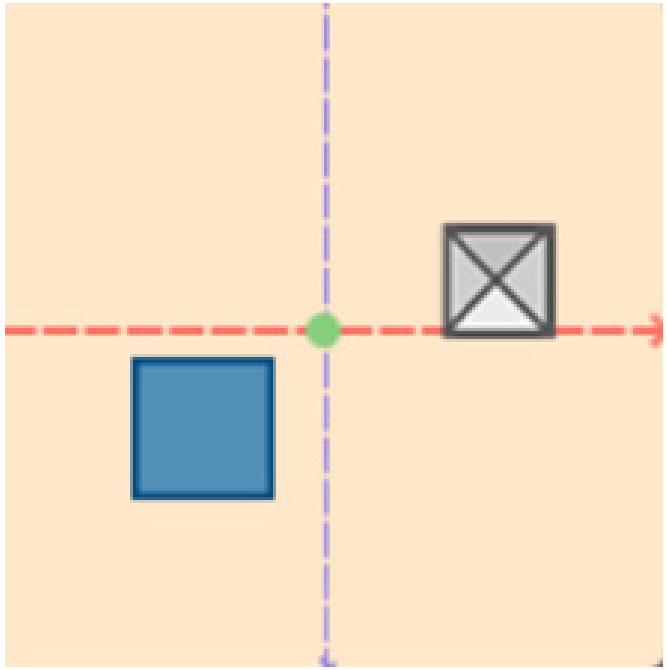
Coordinate spaces

- **World coordinate space**
  - A reference coordinate system that is always fixed
  - Local coordinate can be placed arbitrarily in world coordinate



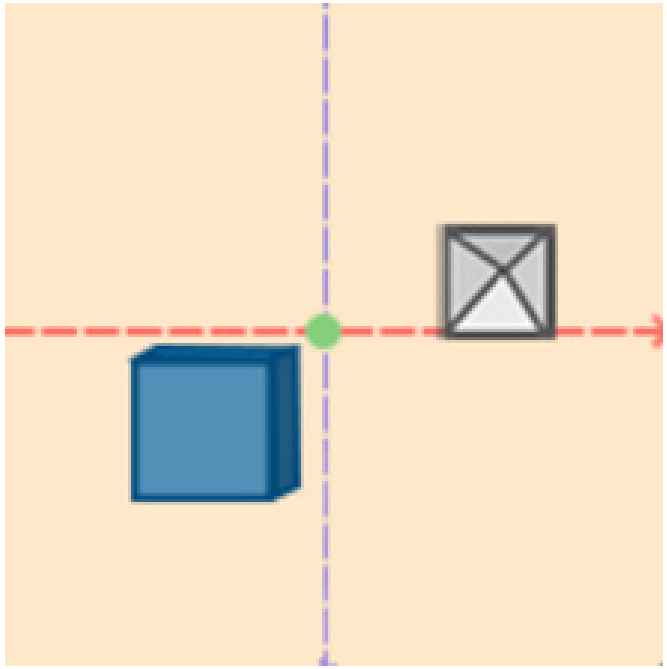
Coordinate spaces

- **View coordinate space**
  - Camera space or eye space
  - Transform world-space coordinates to coordinates that are in front of the user's view (still 3D)



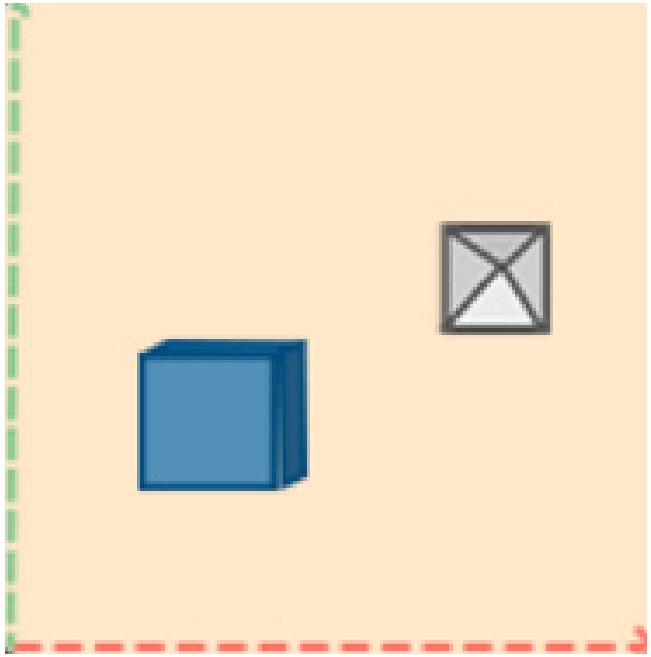
Coordinate spaces

- **Clip coordinate space**
  - Expect the coordinates to be within a specific range
  - Any coordinate that falls outside this range is clipped
  - Projection is done (3D to 2D)



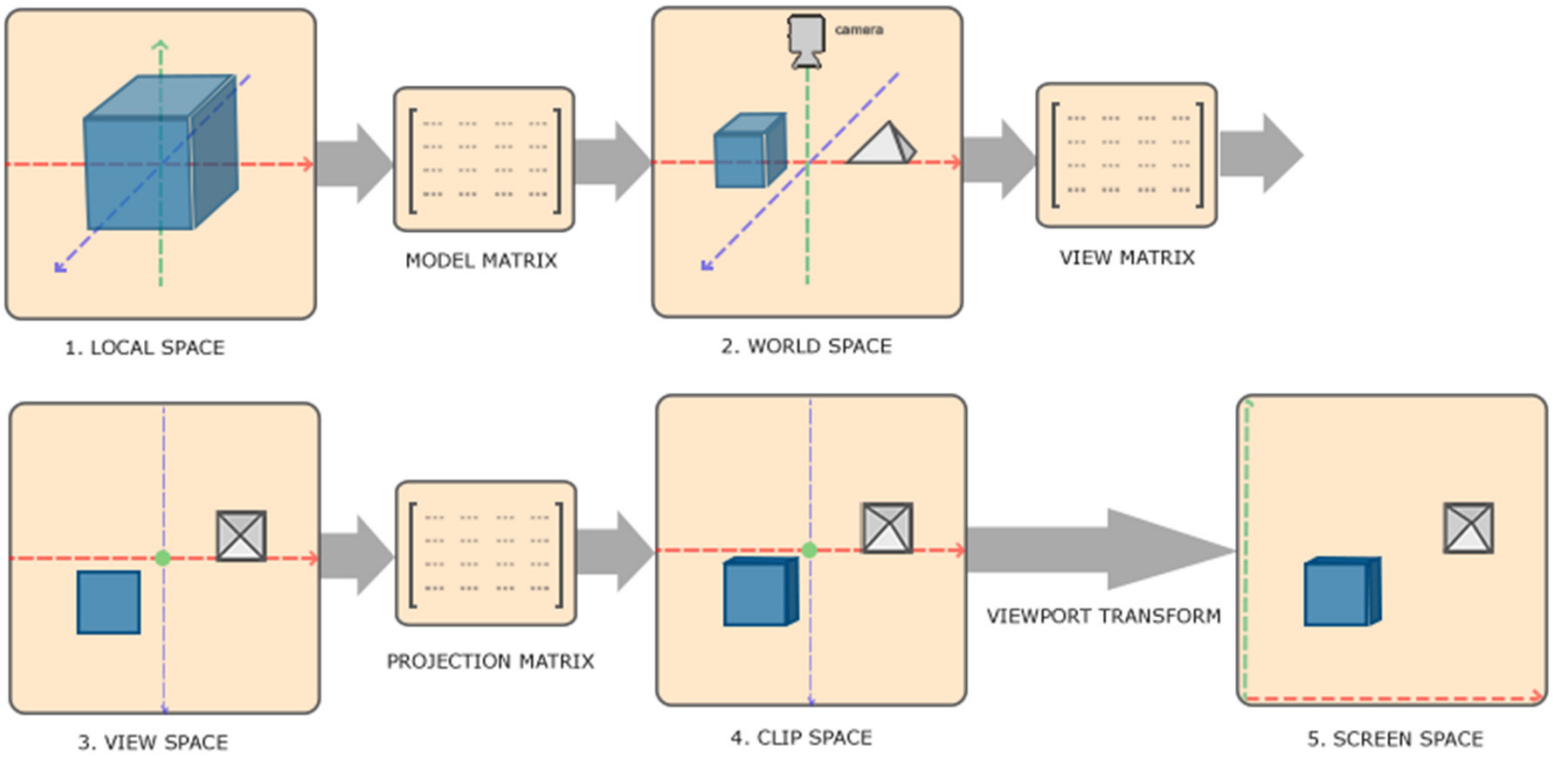
Coordinate spaces

- **Screen coordinate space**
  - The space for display
  - The resulting coordinates are then sent to the rasterizer to turn the continuous representation into fragments/pixels



Coordinate spaces

- **The global picture**
  - Space transformations using matrices



[-1,1]^2

[0,1]^2

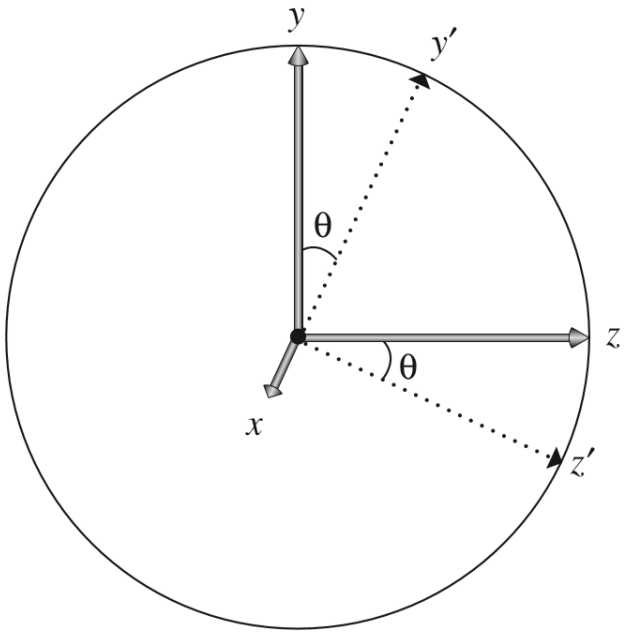
Coordinate transformation

- **Translation transformation**
  - In homogeneous matrix form, the translation transformation is

$$\mathrm{T}(\Delta x, \Delta y, \Delta z) = \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Coordinate transformation

- **Rotation transformation**
  - Rotation about x-coordinate
    - Rotation by an angle θ about the x axis leaves the x coordinate unchanged



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

Positive angle follows the left-hand-side rule from x to z

Coordinate transformation

- **Rotation transformation**
  - Rotation about y- and z-axes

$$\mathbf{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{R}_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

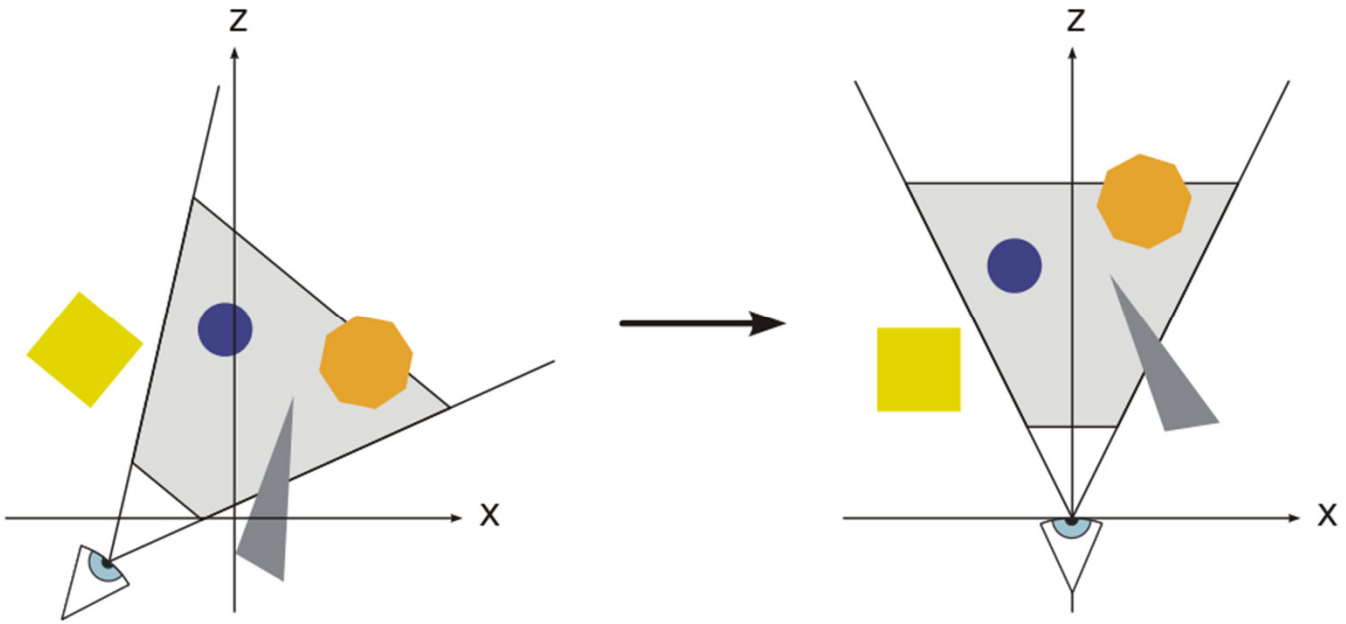
  - An arbitrary rotation can be decomposed into rotations about x-, y- and z-axes

$$\mathbf{R}(\theta) = \mathbf{R}_z(\theta)\,\mathbf{R}_y(\theta)\,\mathbf{R}_x(\theta)$$

4. Projection-based graphics

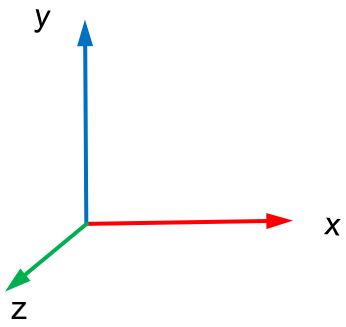
View transformation

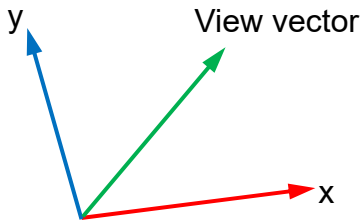
- **What is a view transformation?**
  - Transform the world coordinates into the view (camera/eye) coordinates



View transformation

- **How to compute the view transform?**
  - Translation + rotation from world coordinate system
  - World coordinate system forms an identity matrix
  - Thus, view matrix is formed by camera coordinate system + camera translation in world coordinates
- **View transformation matrix**





$$I = RTB_v$$

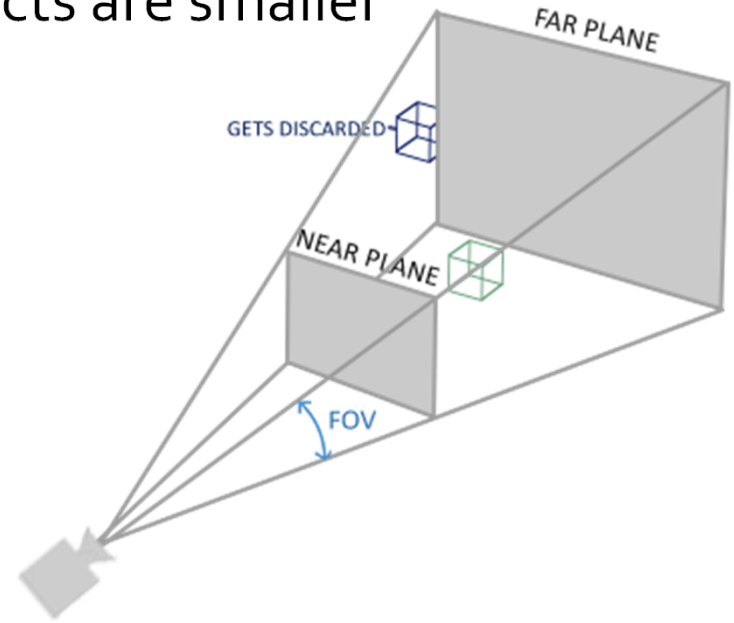
$$B_v = T^{-1}R^T$$

Model-view transformation

- **In practice, we will combine the model transformation and view transformation**
  - Model transformation: determine the final coordinates in world coordinate system
  - View transformation: transform the final world coordinates to view (camera) coordinates
  - Computation:
    - $\mathbf{M}=\mathbf{M}_{view}\,\mathbf{M}_{model}=\mathbf{M}_{view}\,(\ldots\mathbf{S}_{model}\,\mathbf{R}_{model}\,\mathbf{T}_{model})$

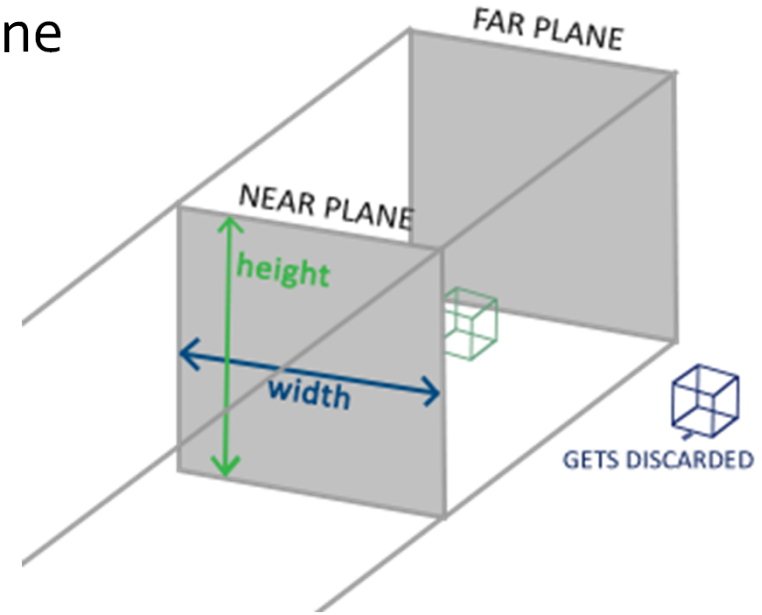
Perspective projection

- **Clipping & projection**
  - A large *frustum* that defines the clipping space
  - All the coordinates inside this frustum is projected along perspective projection line to the projection plane
  - Farther objects are smaller



Orthogonal projection

- **Clipping & projection**
  - A cube-like *frustum* that defines the clipping space
  - All the coordinates inside this frustum is projected along the parallel lines to the projection plane
  - Object sizes do not depend on the distance to the projection plane



Perspective projection representation

- **Perspective projection matrix**
  - Perspective projection for a projection frustum
  - http://www.songho.ca/opengl/gl_projectionmatrix.html

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

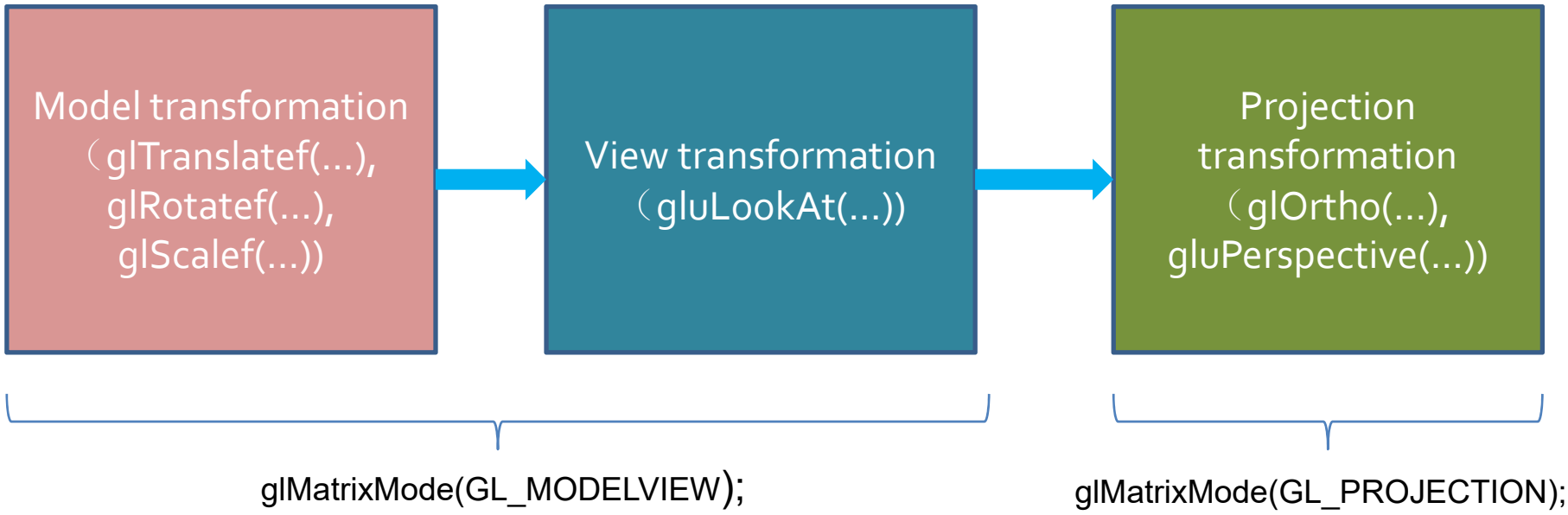
Orthogonal projection representation

- **Orthogonal projection matrix**
  - http://www.songho.ca/opengl/gl_projectionmatrix.html

$$\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

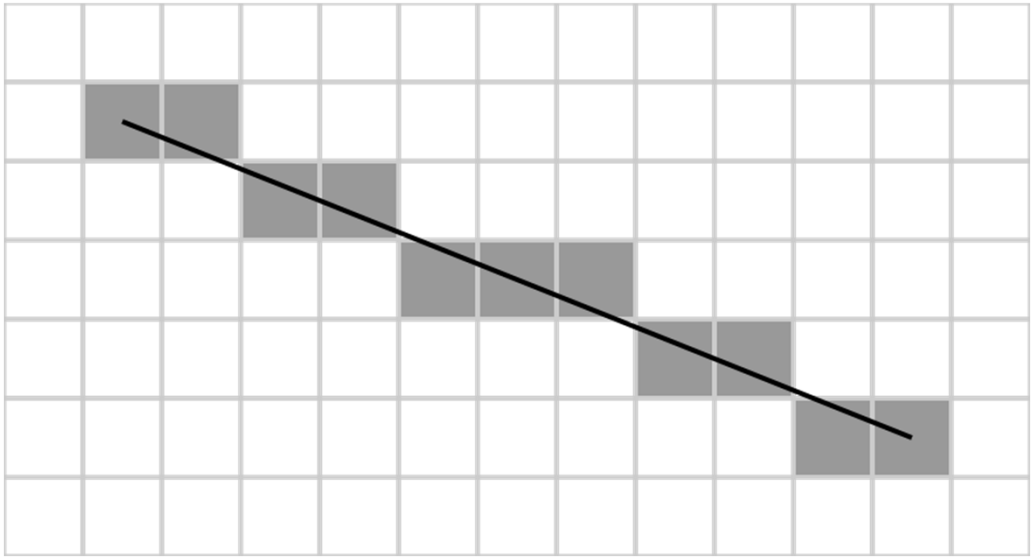
Coordinate transformation in OpenGL

- **The whole transformation**



Line rasterization

- **Bresenham's line algorithm**
  - An algorithm that determines the rasterized points that form a close approximation to a straight line between two end points



Line rasterization

- **Bresenham's line algorithm**
  - Line equation

$$y = mx + b$$
$$y = \frac{(\Delta y)}{(\Delta x)}x + b$$
$$(\Delta x)y = (\Delta y)x + (\Delta x)b$$
$$0 = (\Delta y)x - (\Delta x)y + (\Delta x)b$$

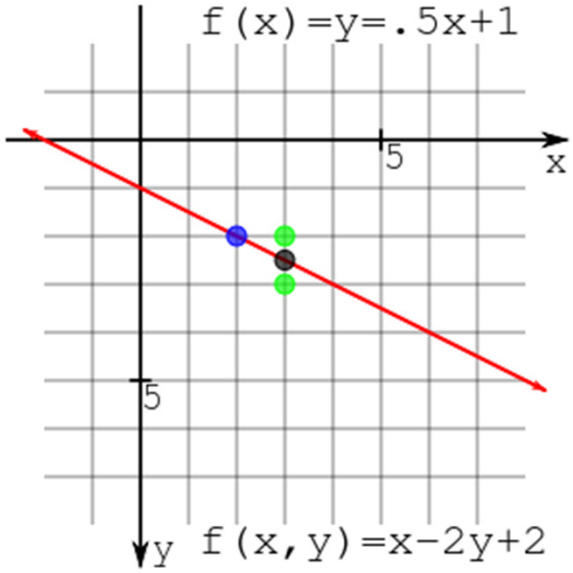
  - Let the last equation be a function of x and y:

$$f(x, y) = 0 = Ax + By + C$$

- $A = \Delta y$
- $B = -\Delta x$
- $C = (\Delta x)b$

Line rasterization

- **Bresenham's line algorithm**
  - Starting from (x_0, y_0), determine the next point to be (x_0+1, y_0) or (x_0+1, y_0+1)
  - Intuition: the point should be chosen based upon which is closer to the line at x_0+1



Evaluate the line function at the midpoint

$$f(x_0 + 1, y_0 + 1/2)$$

f<=0: select (x_0+1, y_0)
otherwise
f>0: select (x_0+1, y_0+1)

Point-inside-polygon test

- **Point-in-triangle test**
  - Compute triangle edge equation from projected positions of vertices

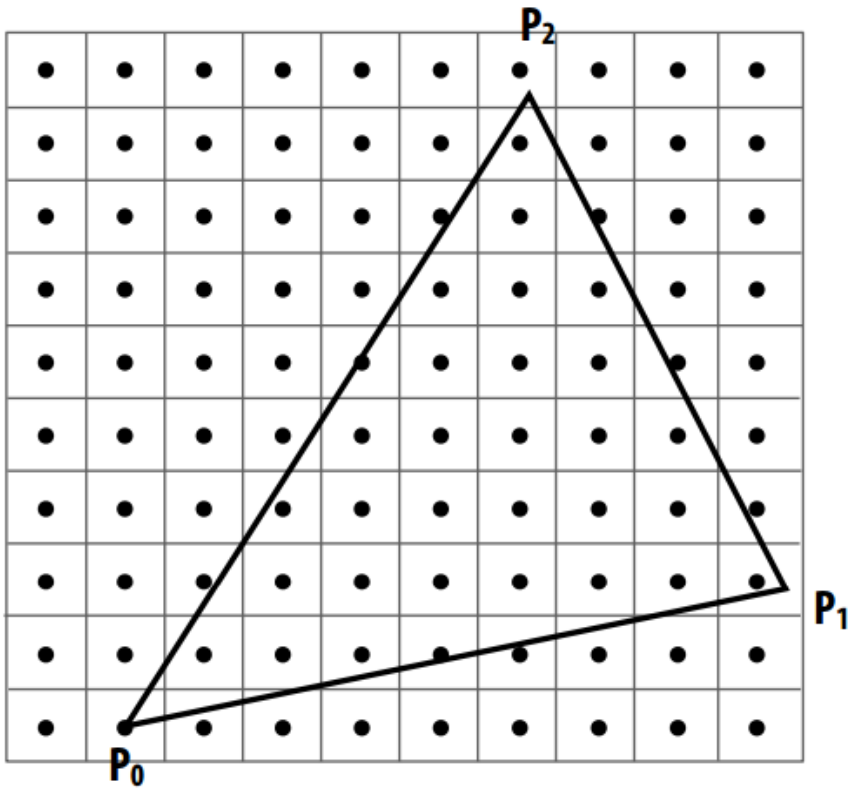
triangle vertices

$$P_i = (X_i, Y_i)$$

$$dX_i = X_{i+1} - X_i$$
$$dY_i = Y_{i+1} - Y_i$$

$$E_i(x, y) = (x - X_i)\, dY_i - (y - Y_i)\, dX_i$$
$$= A_i\, x + B_i\, y + C_i$$

$$E_i(x, y) = 0 \text{ : point on edge}$$
$$> 0 \text{ : outside edge}$$
$$< 0 \text{ : inside edge}$$

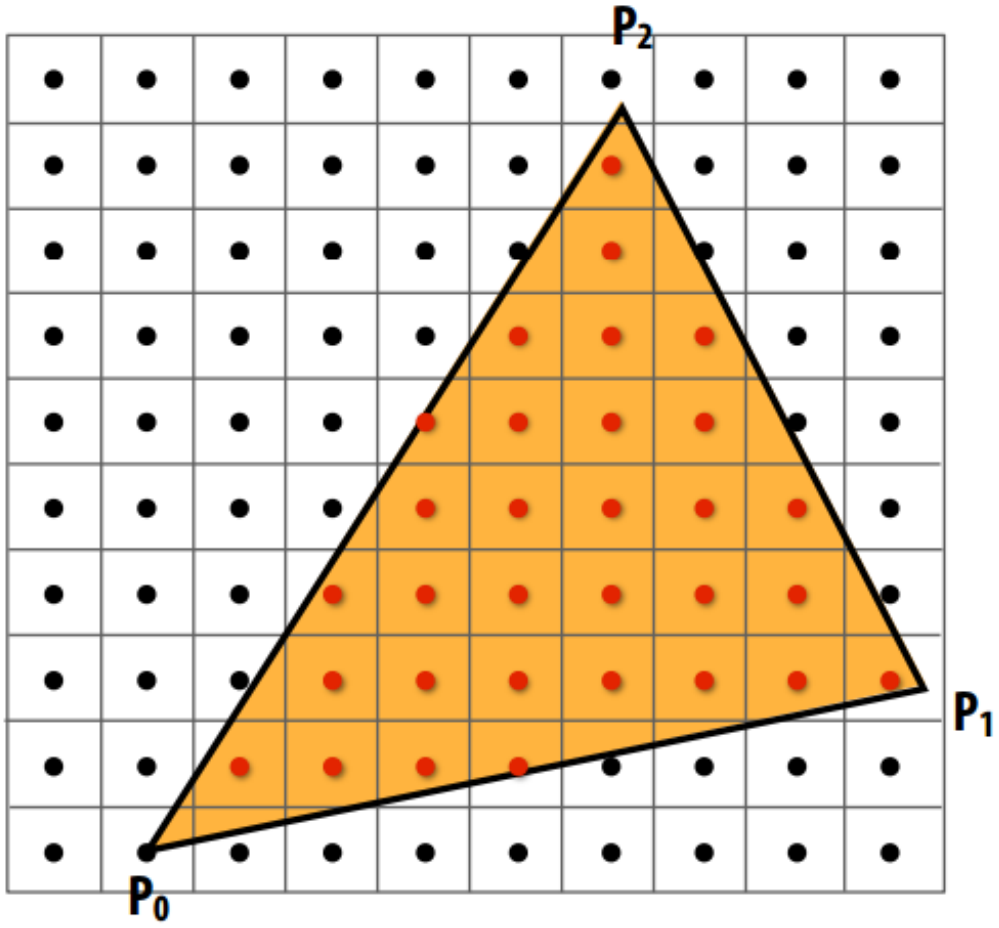


Point-inside-polygon test

Sample point $s = (sx, sy)$ is inside the triangle if it is inside all three edges.

$$
\begin{aligned}
&inside(sx, sy) = \\
&\quad E_0(sx, sy) < 0 \;\&\& \\
&\quad E_1(sx, sy) < 0 \;\&\& \\
&\quad E_2(sx, sy) < 0;
\end{aligned}
$$

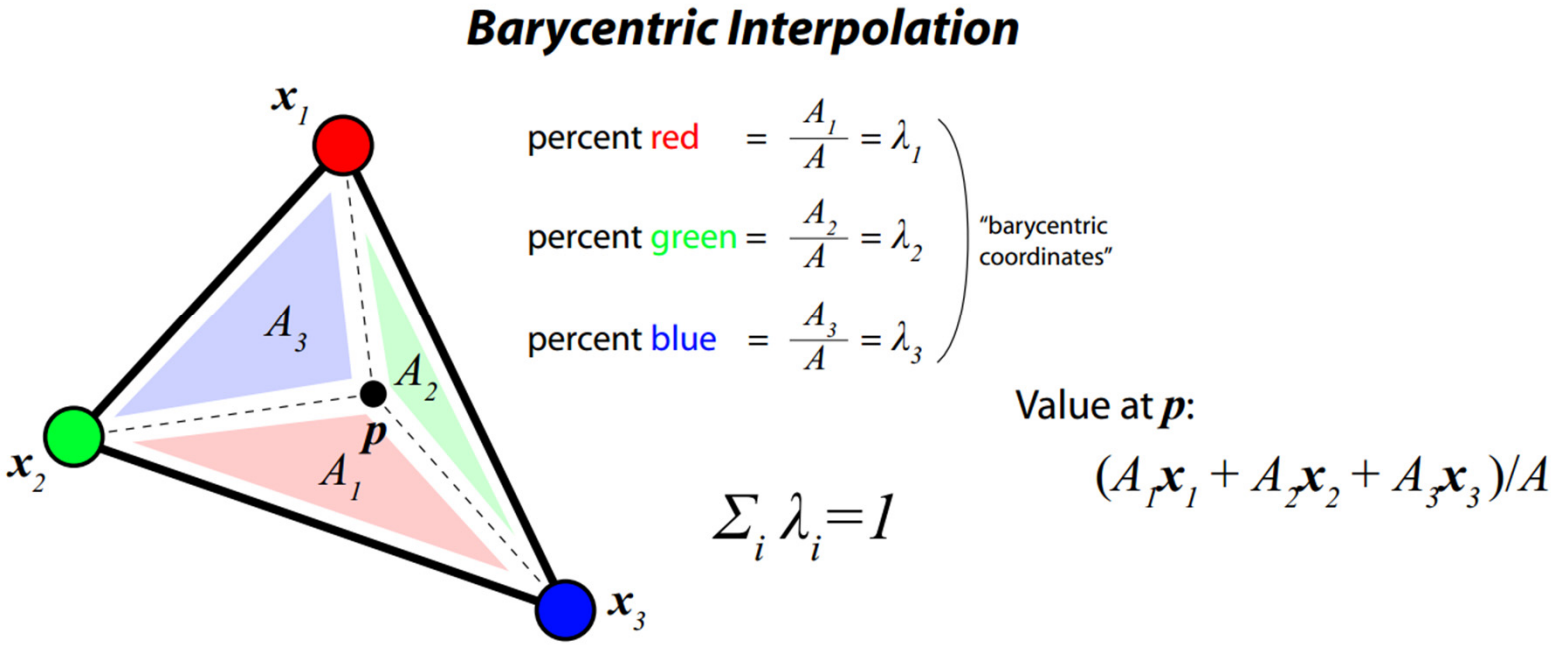
Note: actual implementation of $inside(sx,sy)$ involves ≤ checks based on the triangle coverage edge rules (see beginning of lecture)



Sample points inside triangle are highlighted red.

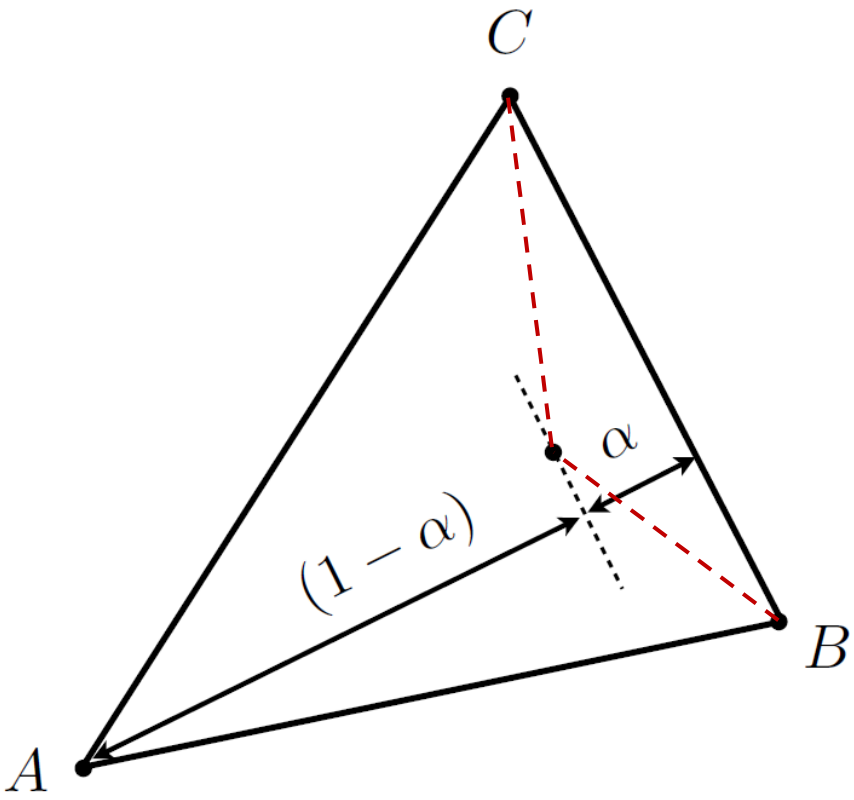
Color interpolation

- **How to fill the color of the pixels inside the triangle region?**
  - Use barycentric interpolation (another approach)



Computing barycentric coordinates

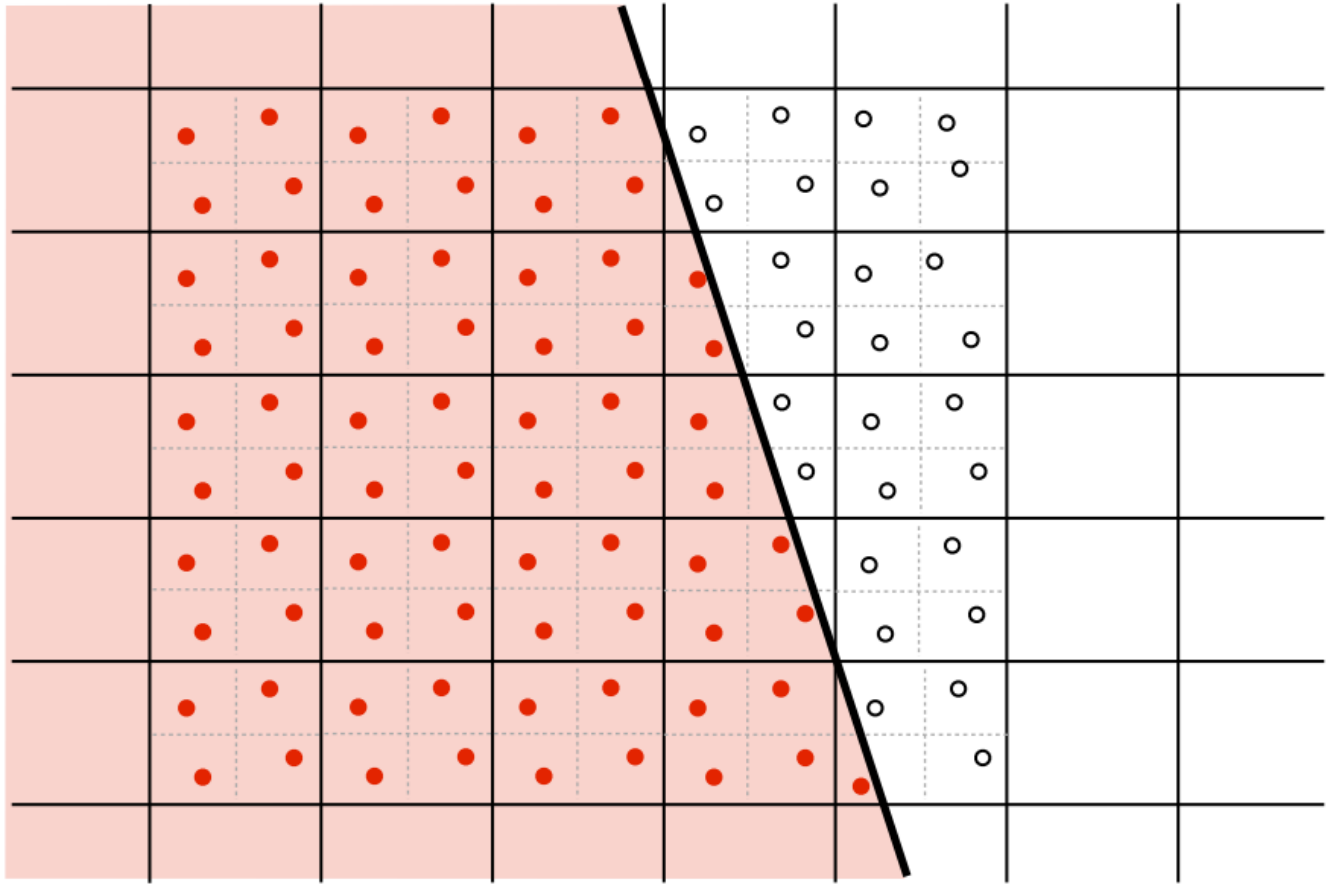
- **Geometric viewpoint — proportional distances**



Similar construction for other coordinates

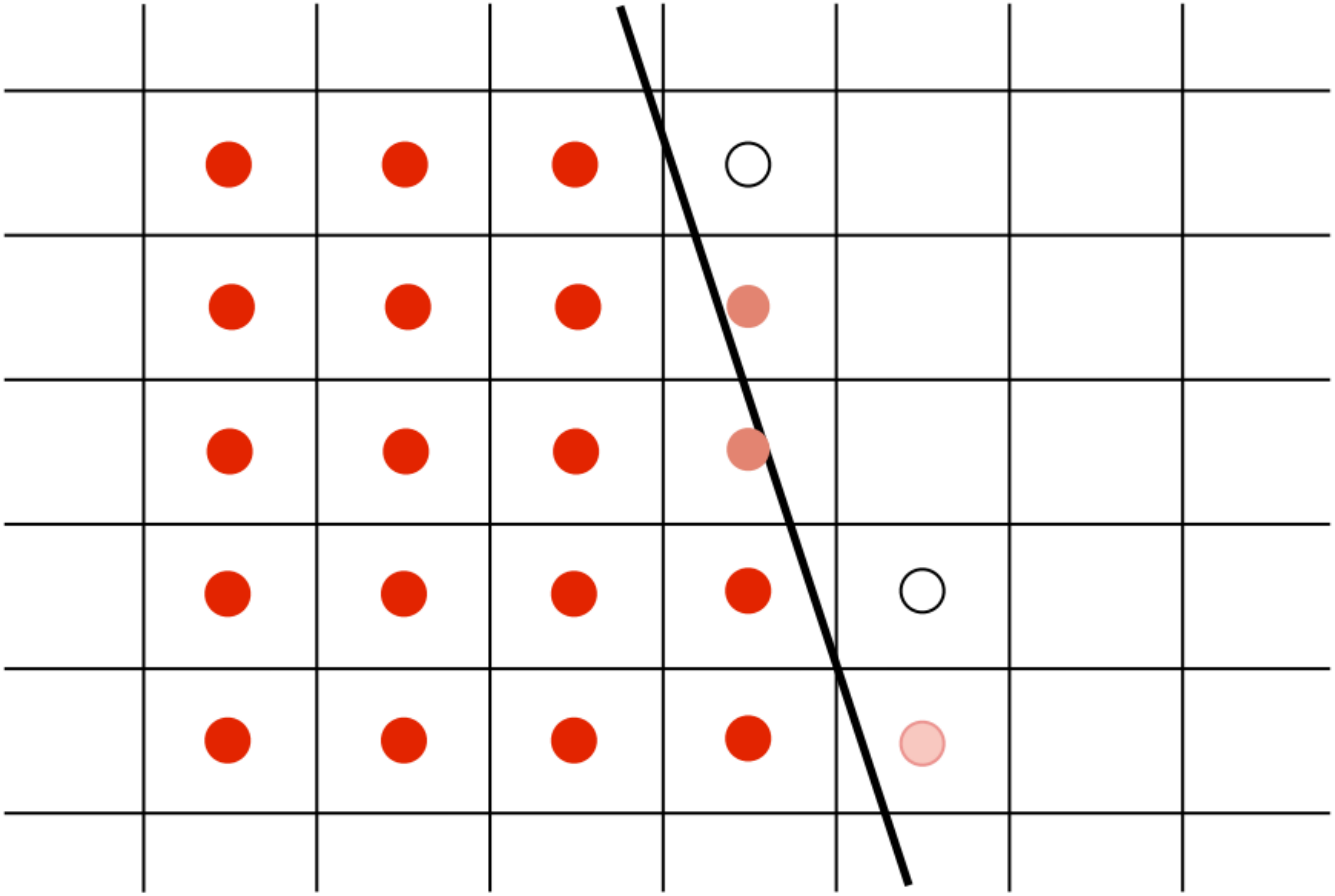
Antialiasing techniques

- **Super-sampling**
  - Example: stratified sampling using four samples per pixel



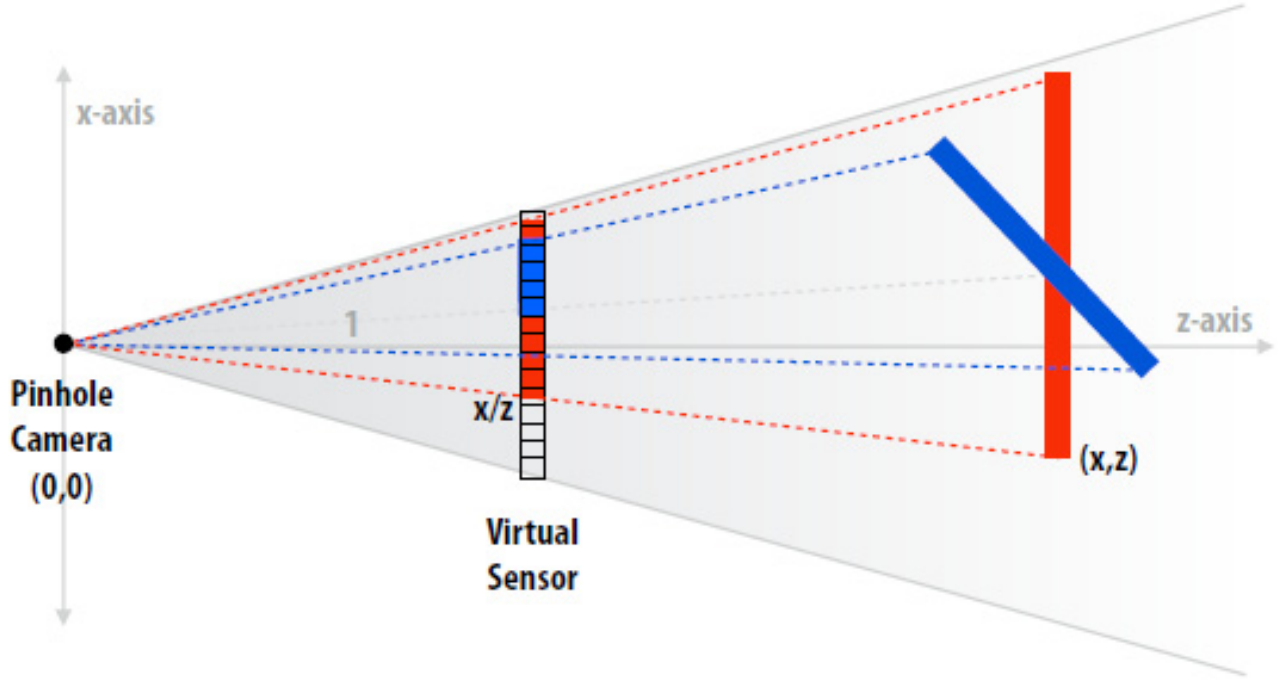
Antialiasing techniques

- **Super-sampling**
  - Resample to display's resolution (box filter)



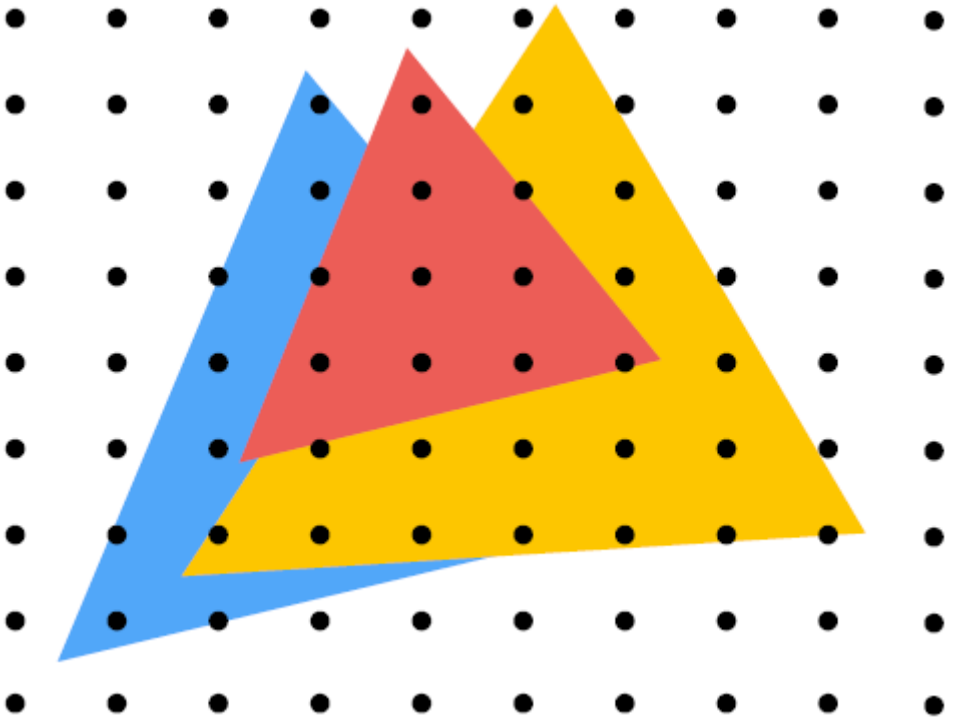
Concept of depth

- The distance of a 3D point to the imaging plane



Visibility

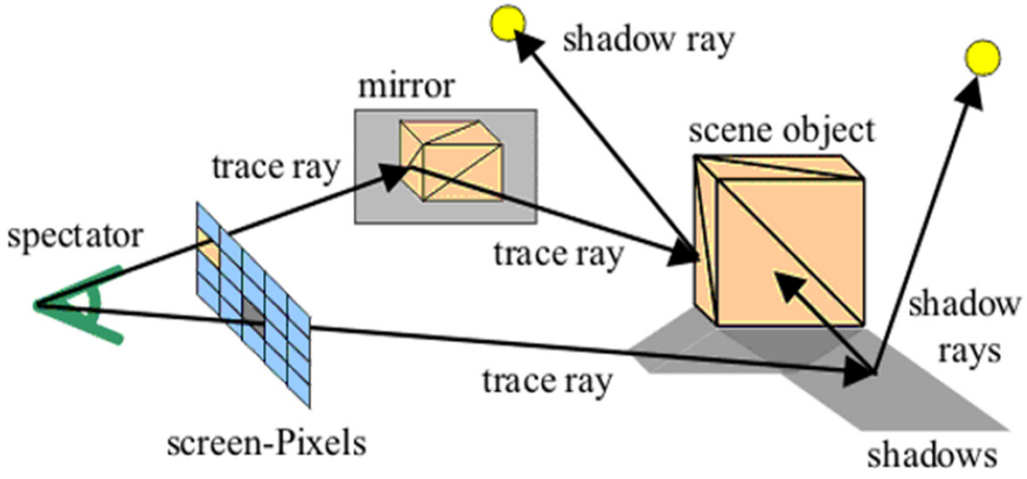
- **Visible surface determination**
  - The process used to determine which surfaces and parts of surfaces are not visible from a certain viewpoint



5. Ray-based graphics

Ray tracing

- **What shall we need for ray tracing?**
  - A set of rays shooting from imaging plane
  - Light source distribution
  - Ray-object intersection
    - Normal, texture coordinates
    - Reflected and refracted rays
  - How the object reflects light

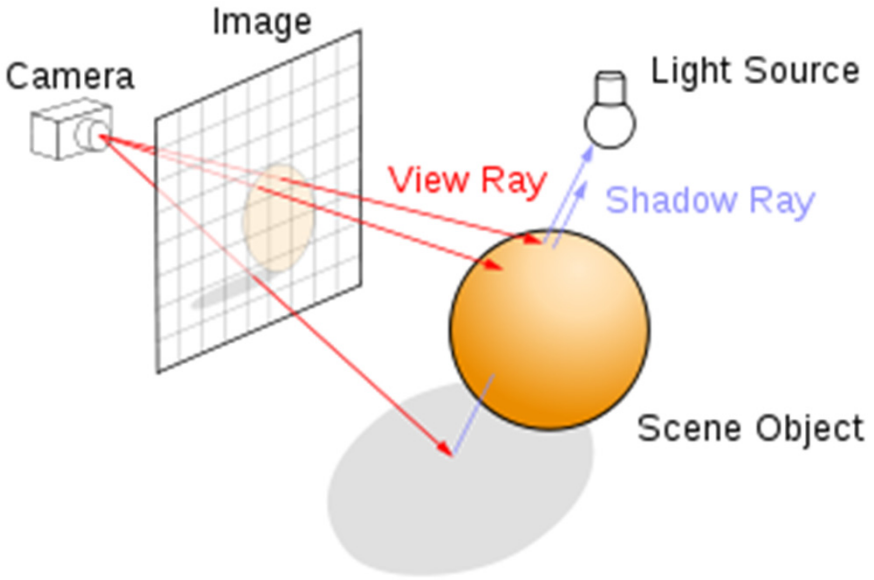


Optical ray

- **Shooting optical rays from focal point and through each pixel in imaging plane**
  - Rays are generated by connecting focal point and image pixel center

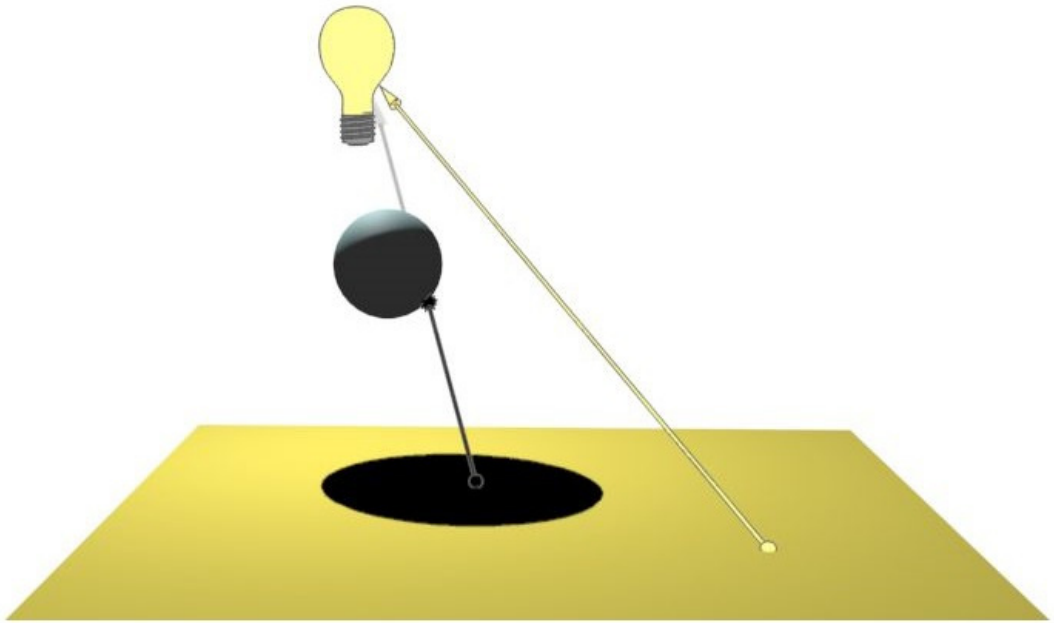
- **Ray expression**

$$\mathrm{r}(t) = \mathrm{o} + t\mathbf{d} \qquad 0 \le t \le \infty$$



Shadow rendering

- **At each intersection point**
  - A shadow ray is generated towards light source
  - If intersection with other objects, the point is inside shadow region



Ray-triangle intersection

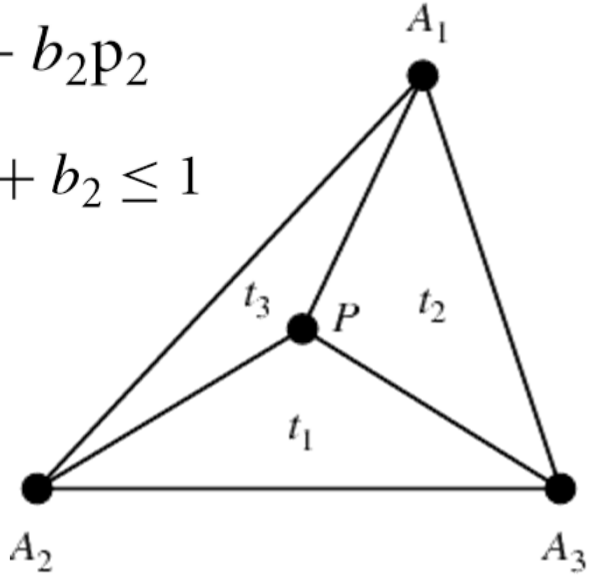
- **An efficient ray-triangle intersection algorithm?**
  - Can also be derived using barycentric coordinates
- **For triangle**
  - Any point inside the triangle can be written as:

$$\mathrm{p}(b_1, b_2) = (1 - b_1 - b_2)\mathrm{p}_0 + b_1\mathrm{p}_1 + b_2\mathrm{p}_2$$

With conditions: $b_1 \geq 0, b_2 \geq 0$ $b_1 + b_2 \leq 1$

  - Insert parametric ray equation

$$\mathrm{o} + t\mathbf{d} = (1 - b_1 - b_2)\mathrm{p}_0 + b_1\mathrm{p}_1 + b_2\mathrm{p}_2$$



Ray-triangle intersection

- **Equation to solve**

$$\begin{pmatrix} -\mathbf{d} & \mathbf{e}_1 & \mathbf{e}_2 \end{pmatrix} \begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \mathbf{s}$$

- **How to solve such an equation?**
  - Cramer's rule

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{|\,-\mathbf{d} \quad \mathbf{e}_1 \quad \mathbf{e}_2\,|} \begin{bmatrix} |\,\mathbf{s} \quad \mathbf{e}_1 \quad \mathbf{e}_2\,| \\ |\,-\mathbf{d} \quad \mathbf{s} \quad \mathbf{e}_2\,| \\ |\,-\mathbf{d} \quad \mathbf{e}_1 \quad \mathbf{s}\,| \end{bmatrix}$$

we write $|\,\mathbf{a} \quad \mathbf{b} \quad \mathbf{c}\,|$ to mean the determinant

Ray-triangle intersection

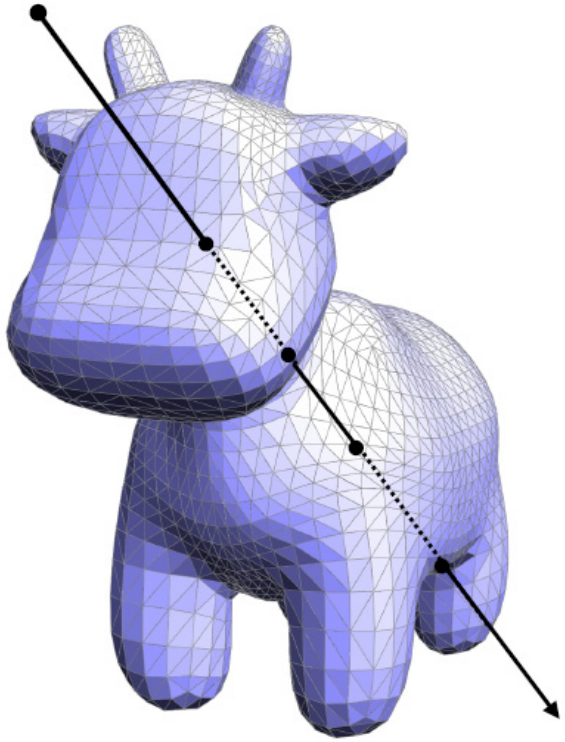
- **Is this solver efficient?**
  - No!
- **More observation**
  - Determinant identify in 3D

$$|\,\mathbf{a}\quad\mathbf{b}\quad\mathbf{c}\,| = -(\mathbf{a}\times\mathbf{c})\cdot\mathbf{b} = -(\mathbf{c}\times\mathbf{b})\cdot\mathbf{a}$$

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{(\mathbf{d}\times\mathbf{e}_2)\cdot\mathbf{e}_1}\begin{bmatrix} (\mathbf{s}\times\mathbf{e}_1)\cdot\mathbf{e}_2 \\ (\mathbf{d}\times\mathbf{e}_2)\cdot\mathbf{s} \\ (\mathbf{s}\times\mathbf{e}_1)\cdot\mathbf{d} \end{bmatrix} \Rightarrow \begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\mathbf{s}_1\cdot\mathbf{e}_1}\begin{bmatrix} \mathbf{s}_2\cdot\mathbf{e}_2 \\ \mathbf{s}_1\cdot\mathbf{s} \\ \mathbf{s}_2\cdot\mathbf{d} \end{bmatrix}$$

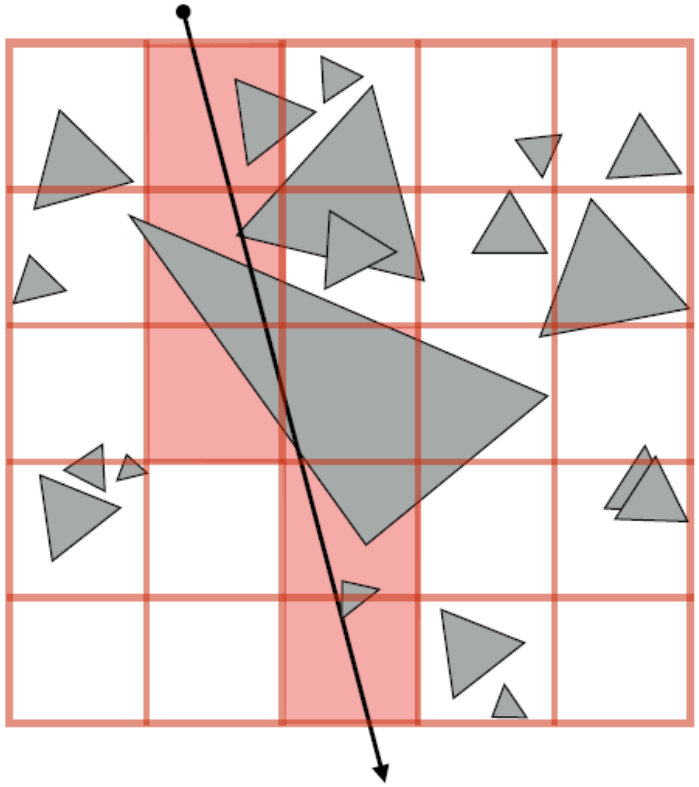
Ray-mesh intersection

- **How to intersect a mesh?**
  - Intersect its triangles
  - Search the triangles the ray hits
  - Obtain intersection point and normal



Ray-mesh intersection

- **How to search the intersected triangles?**
  - Linear search -> too slow
  - Grid acceleration structure
    - For each cell, we record 2D triangles included



- Partition space into equal sized volumes ("voxels")
- Each grid cell contains primitives that overlap voxel. (very cheap to construct acceleration structure)
- Walk ray through volume in order
  - Very efficient implementation possible (think: 3D line rasterization)
  - Only consider intersection with primitives in voxels the ray intersects

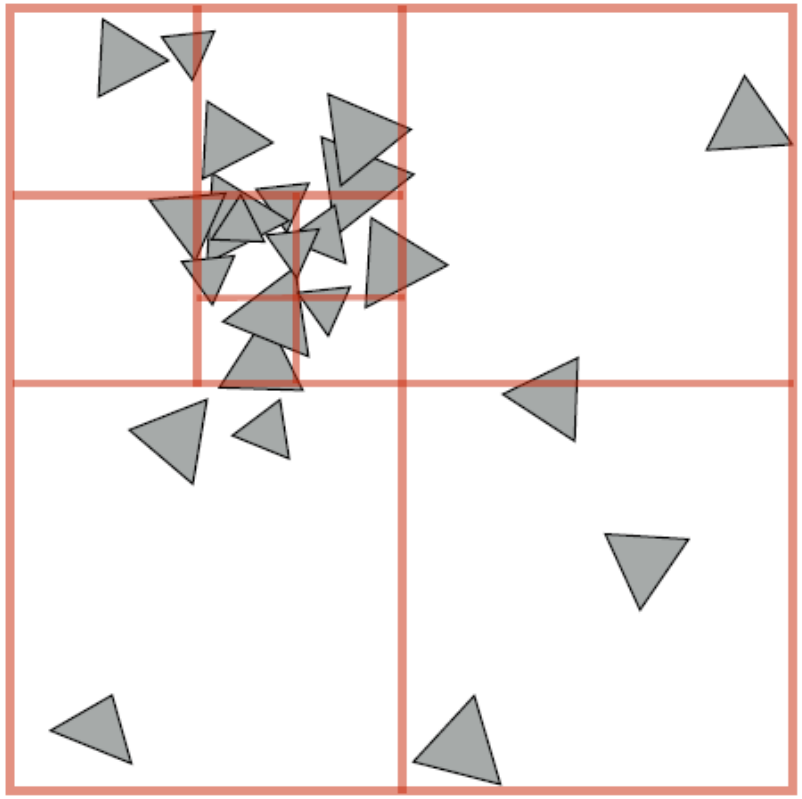
Ray-mesh intersection

- **Quad-tree / octree**

  Like uniform grid: easy to build (don't have to choose partition planes)

  Has greater ability to adapt to location of scene geometry than uniform grid.

  But lower intersection performance than K-D tree (only limited ability to adapt)

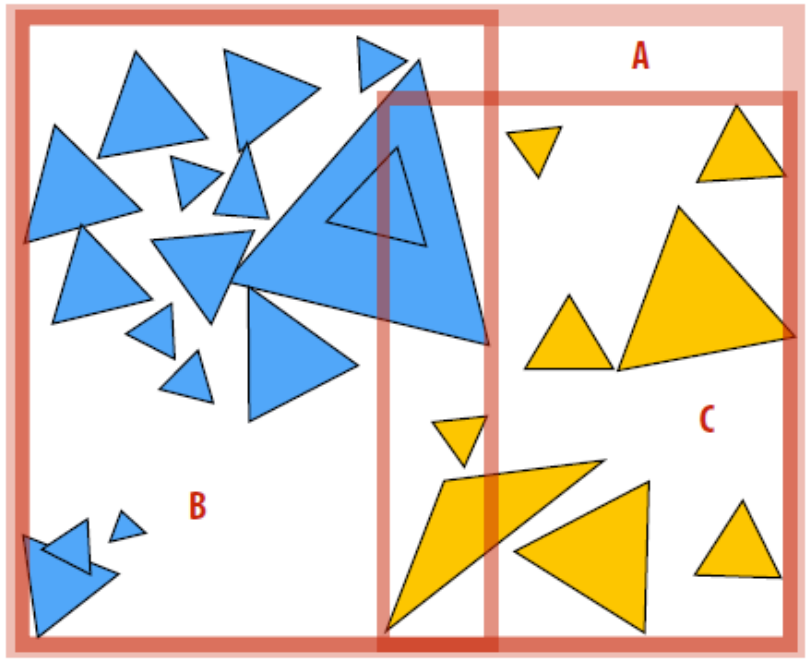


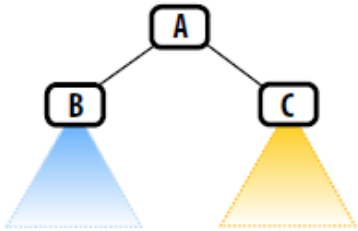
Quad-tree: nodes have 4 children (partitions 2D space)

Octree: nodes have 8 children (partitions 3D space)

Bounding volume hierarchies

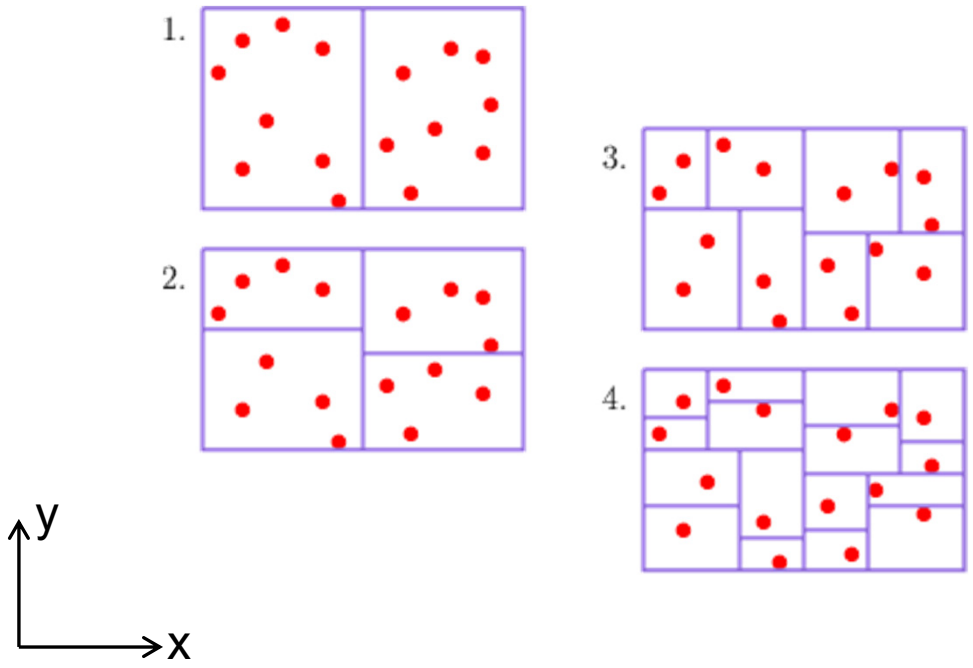
- Another example
  - BVH partitions each node's primitives into disjoints sets
    - Note: The sets can still be overlapping in space (below: child bounding boxes may overlap in space)





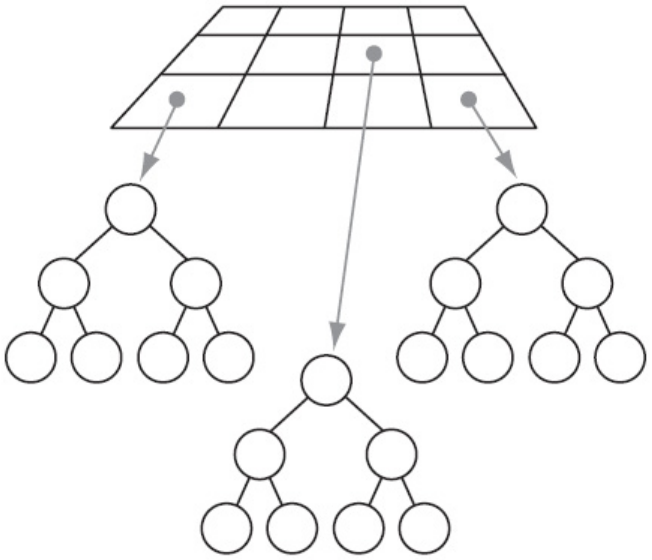
K-D tree construction

- **Primitive representation**
  - Each primitive is represented by its centroid
- **Coordinate cycling**

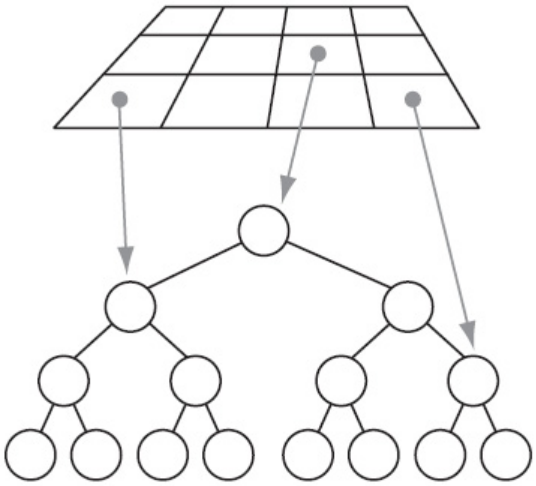


Hybrid Spatial Partitioning

- **Hybrid schemes**
  - A grid of trees, each grid cell containing a separate tree or part of a tree



(a)



(b)

Anti-aliasing

- **Discrepancy definition**
  - A volume in n-dimension
  - A sequence of sample points
  - The discrepancy of P with respect to B is:

$$D_N(B, P) = \sup_{b \in B} \left| \frac{\sharp\{x_i \in b\}}{N} - \lambda(b) \right|$$

$$B = \{[0, v_1] \times [0, v_2] \times \cdots \times [0, v_s]\}$$

$$P = x_1, \ldots, x_N$$

$\sharp\{x_i \in b\}$ is the number of points in b

$\lambda(b)$ is the volume of b

Anti-aliasing

- **Super-sampling**
  - Sampling method
    - Rotated grid : A 2×2 grid layout is used rotated to avoid sample alignment on the axes
    - Greatly improving anti-aliasing quality for the most commonly encountered cases
    - For an optimal pattern, the rotation angle is arctan (1/2) (about 26.6°)

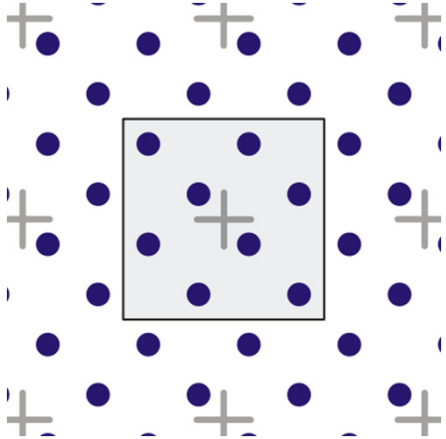


Image Reconstruction

- **Filter functions**
  - Gaussian filter
    - Give a reasonably good result in practice
    - Tend to cause slight blurring of the final image compared to some of the other filters

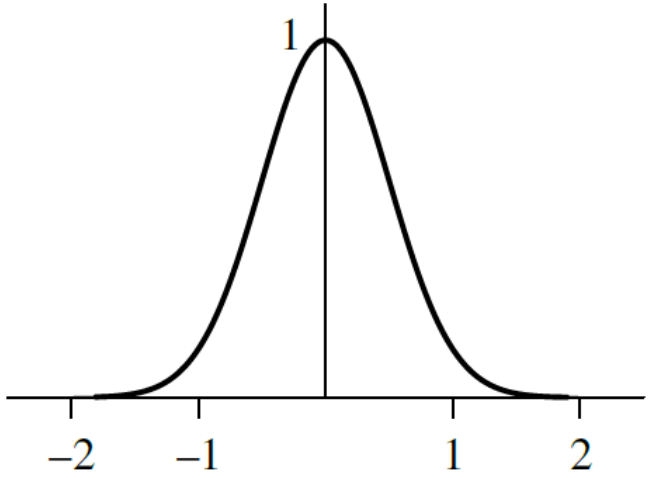
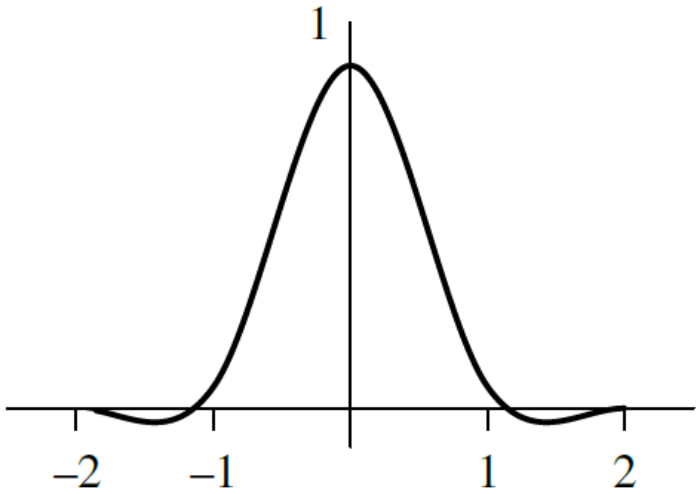


Image reconstruction

- **Filter functions**
  - Mitchell filter
    - Trade-off between *ringing* and *blurring*
    - This filter function takes on negative values, which improve the sharpness of the edges
    - Final pixel values can therefore become negative: clamping

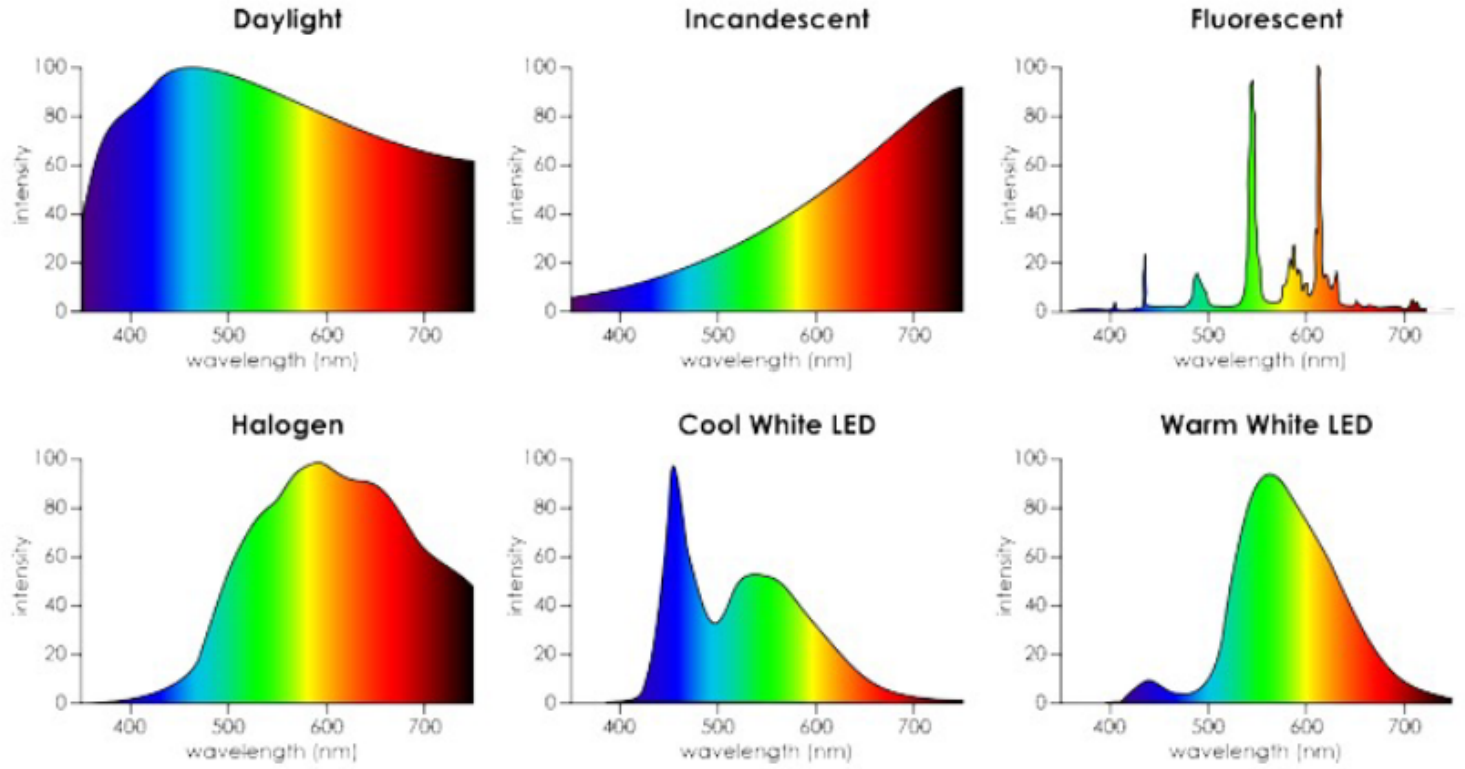


6. Illumination

Power distribution of light

- **Describe the distribution of power (energy/time) by wavelength**

Below: spectrum of various common light sources:



CIE 1931 XYZ primaries

- **Primaries chosen so that convex combination of primaries includes all observable colors (color matching functions for these primaries are non-negative)**

For any spectrum $\Phi(\lambda)$, can express spectrum as weighted combination of primaries:

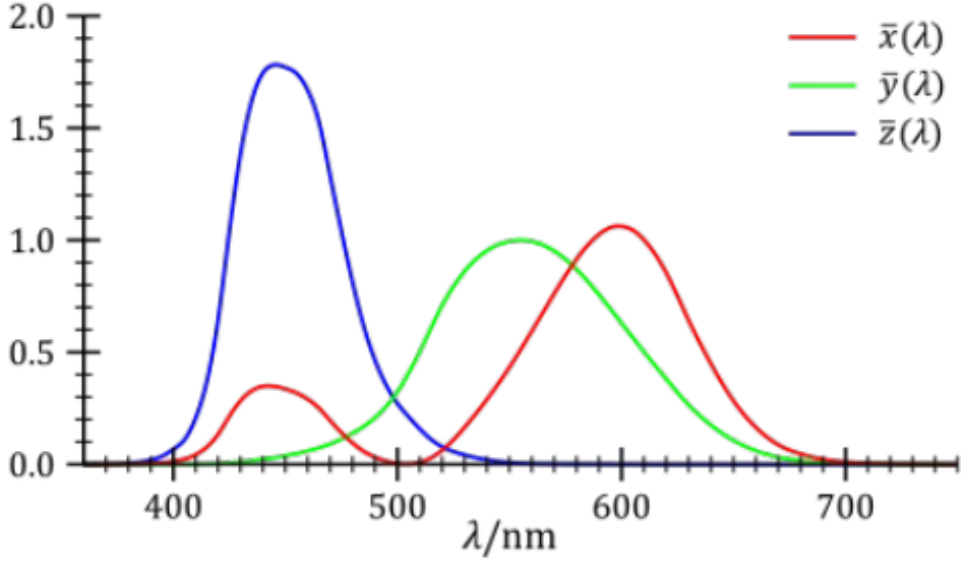
$$X\mathbf{X} + Y\mathbf{Y} + Z\mathbf{Z}$$

$$X = k \int_\lambda \Phi(\lambda)\overline{x}(\lambda)d\lambda$$

$$Y = k \int_\lambda \Phi(\lambda)\overline{y}(\lambda)d\lambda$$

$$Z = k \int_\lambda \Phi(\lambda)\overline{z}(\lambda)d\lambda$$

1931 CIE $\overline{x}, \overline{y}, \overline{z}$ color-matching functions

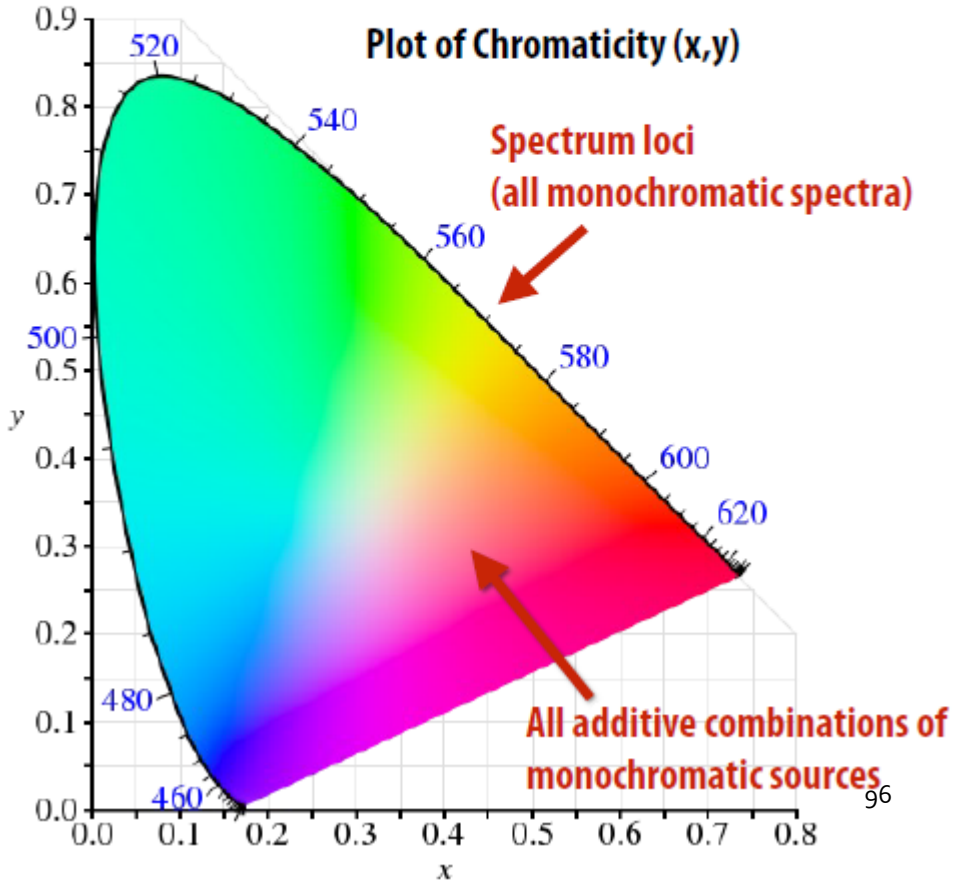


- **What do the XYZ primaries look like? Unlike previous example where primaries were monochromatic R, G, B monochromatic lights, XYZ primaries do not correspond to any physically realizable light sources**

Intensity-independent representation

- Chromaticity is the intensity-independent component of a color
- Project (X,Y,Z) to X+Y+Z=1 plane in XYZ space

$$x = \frac{X}{X+Y+Z}$$
$$y = \frac{Y}{X+Y+Z}$$
$$z = \frac{Z}{X+Y+Z} = 1 - x - y$$



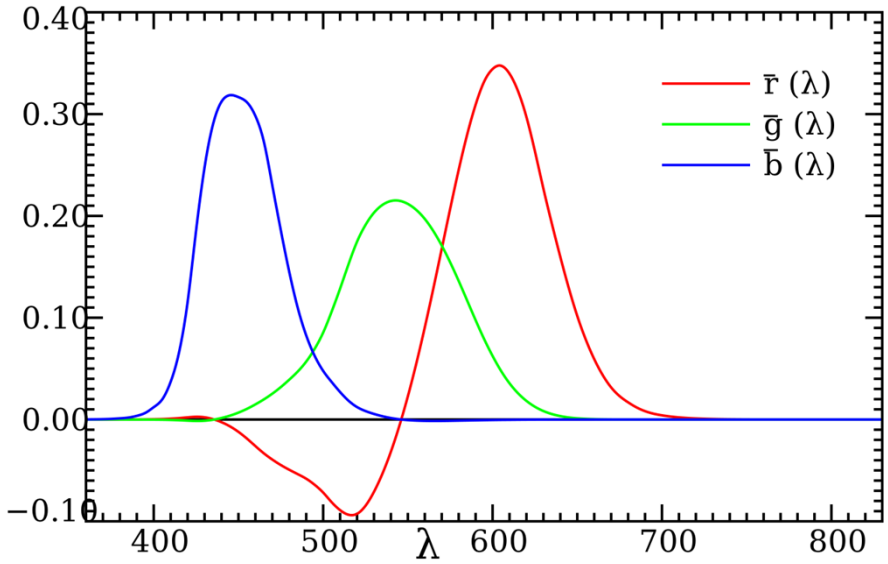
Can recover (X, Y, Z) from (x, y, Y)

$$X = \frac{x}{y}Y$$
$$Y = Y$$
$$Z = \frac{1-(x+y)}{y}Y$$

RGB color

- **Different set of matching functions**

$$\int_0^\infty \bar{r}(\lambda)\,d\lambda = \int_0^\infty \bar{g}(\lambda)\,d\lambda = \int_0^\infty \bar{b}(\lambda)\,d\lambda$$



$$R = \int_0^\infty S(\lambda)\,\bar{r}(\lambda)\,d\lambda$$

$$G = \int_0^\infty S(\lambda)\,\bar{g}(\lambda)\,d\lambda$$

$$B = \int_0^\infty S(\lambda)\,\bar{b}(\lambda)\,d\lambda$$

XYZ-RGB conversions

- **Conversion between XYZ and RGB colors**

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 0.418,47 & -0.158,66 & -0.082,835 \\ -0.091,169 & 0.252,43 & 0.015,708 \\ 0.000,920,90 & -0.002,549,8 & 0.178,60 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

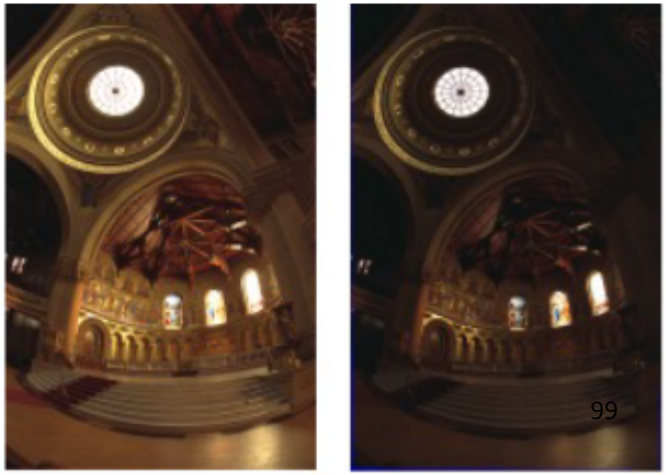
$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{0.176,97} \begin{bmatrix} 0.490,00 & 0.310,00 & 0.200,00 \\ 0.176,97 & 0.812,40 & 0.010,630 \\ 0.000,0 & 0.010,000 & 0.990,00 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

High dynamic range image

- Problem: ratio of brightness object to darkest object in real-world scenes can be quite large
  - Human eye can discern ratio of 100,000:1 (even more if accounting for adaptation)
- High-dynamic range (HDR) image: encodes large range of luminance (or lightness) values
  - Common format: 16-bits per channel EXR (see environment maps in Asst. 3)
- Modern camera senses can only sense much narrower range of luminances (e.g., 12-bit pixels)
- But most modern displays can only display a much narrower range of luminances
  - Luminance of white pixel / luminance of black pixel for a high-end LCD TV ~ 3000:1 *

Overexposed (loss of detail in brightest areas since they are clamped to 1)

Underexposed (detail remains in brightest areas, but large regions of image clamped to 0)



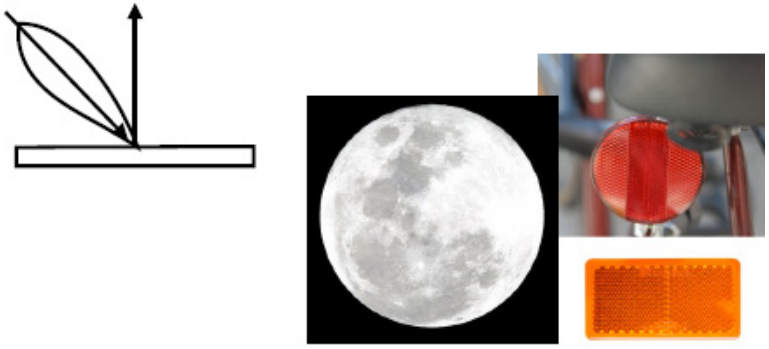
Categories of different materials

- **Ideal specular**
  Perfect mirror

- **Ideal diffuse**
  Uniform reflection in all directions

- **Glossy specular**
  Majority of light distributed in reflection direction

- **Retro-reflective**
  Reflects light back toward source

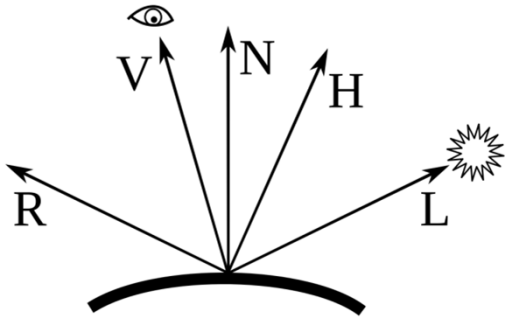


Diagrams illustrate how incoming light energy from given direction is reflected in various directions.

Phong reflection

- **Phong lighting model**
  - Constant ambient lighting
  - Diffuse lighting
  - Specular lighting

$$I_\mathrm{p} = k_\mathrm{a} i_\mathrm{a} + \sum_{m \in \text{ lights}} (k_\mathrm{d} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_\mathrm{s} (\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}})$$

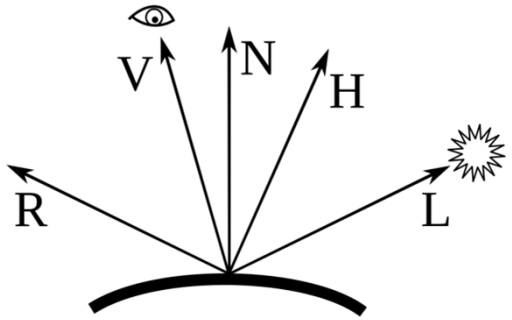


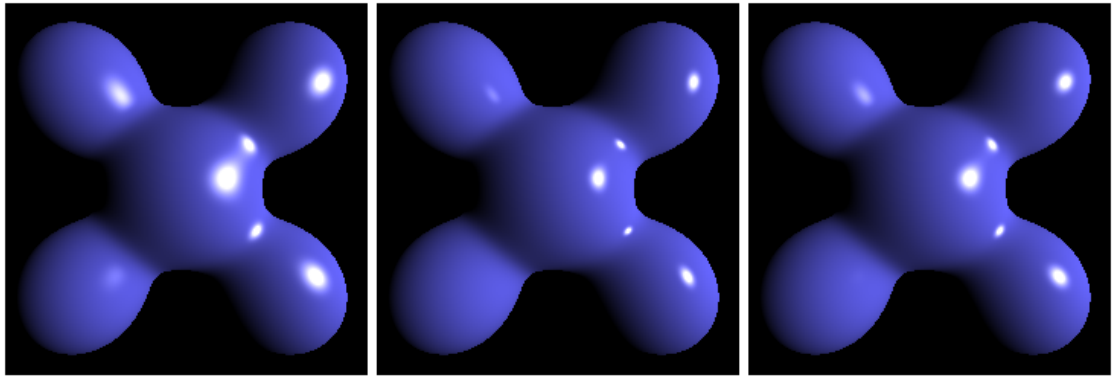
Phong reflection

- **Approximation**
  - Blinn–Phong lighting model

$$H = \frac{L+V}{\|L+V\|}$$

$$(N \cdot H)^{\alpha'} \xrightarrow{\text{replace}} (R \cdot V)^{\alpha}$$

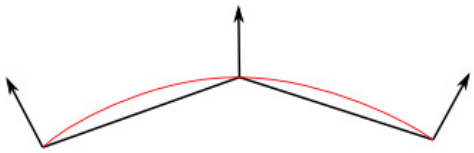


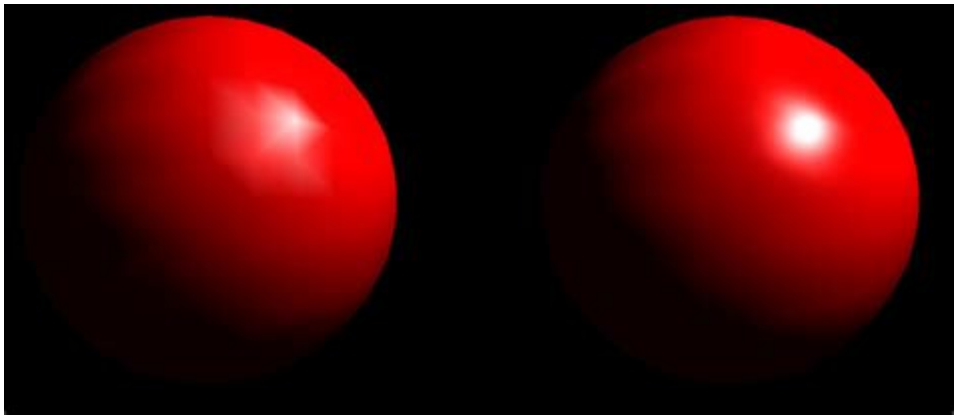


Blinn–Phong | Phong | Blinn–Phong (higher exponent)

Lighting in OpenGL

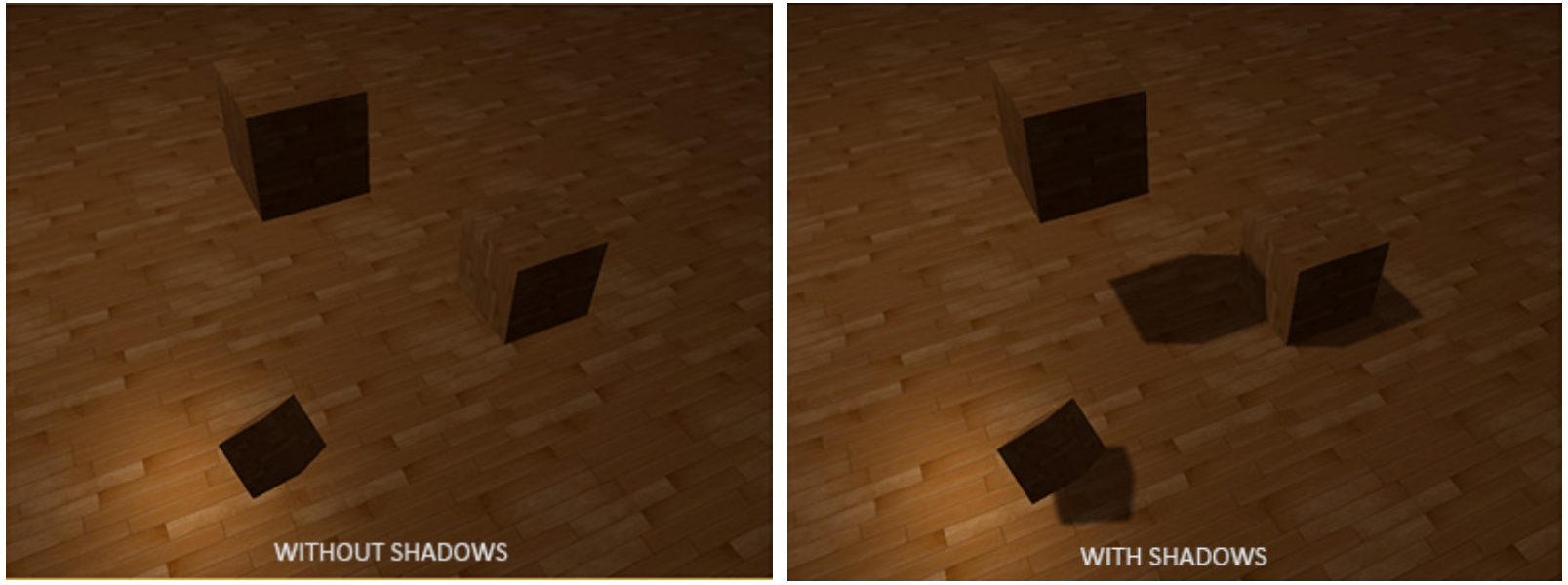
- **Gourand vs. Phong shading**
  - Interpolate colors vs. interpolate normals
  - Gourand shading: colors interpolated within primitives
  - Phong shading: normals are interpolated first per pixel, then colors are computed based on a lighting model





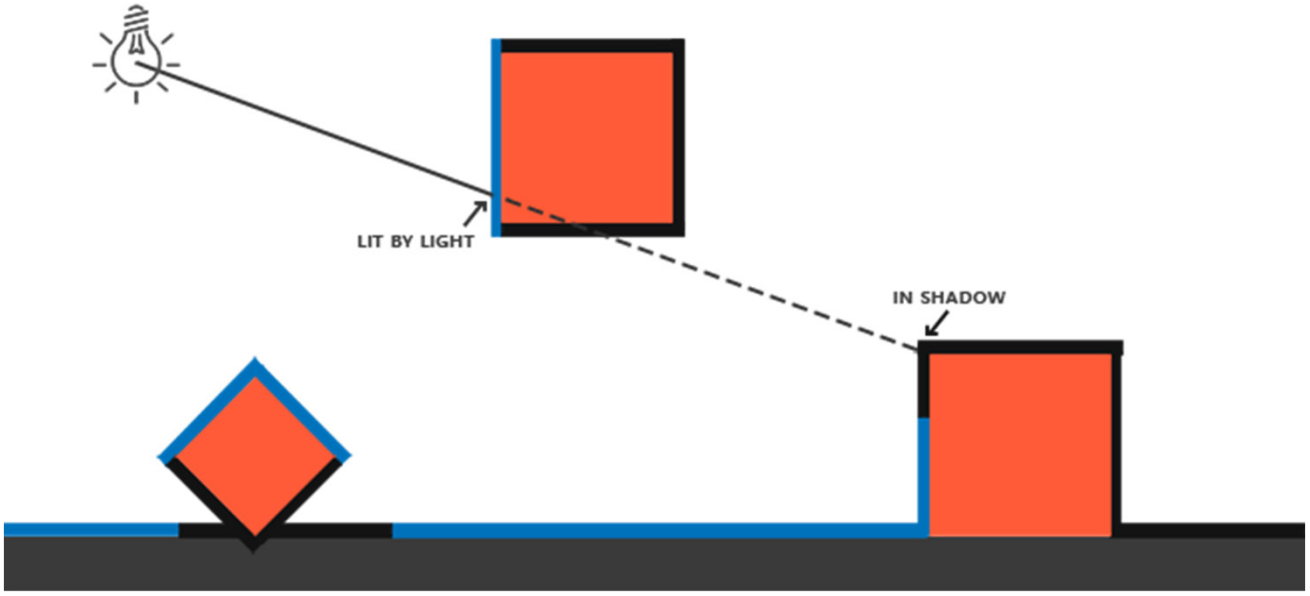
Shadow

- **What is shadow**
  - A result of the absence of light due to occlusion
  - when a light source's light rays do not hit an object because it gets occluded by some other object the object is in shadow



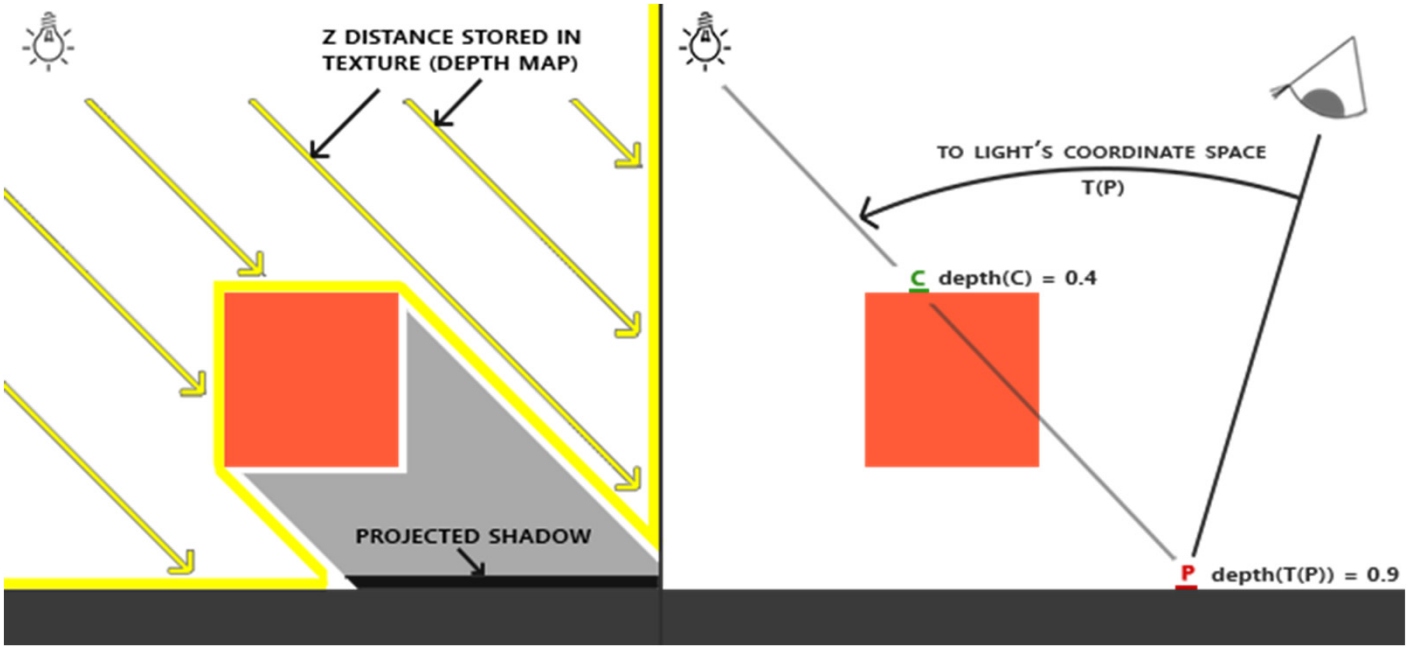
Shadow mapping

- **Basic idea**
  - We render the scene from the light's point of view
  - Everything we see from the light's perspective is lit and everything we can't see must be in shadow



Shadow mapping

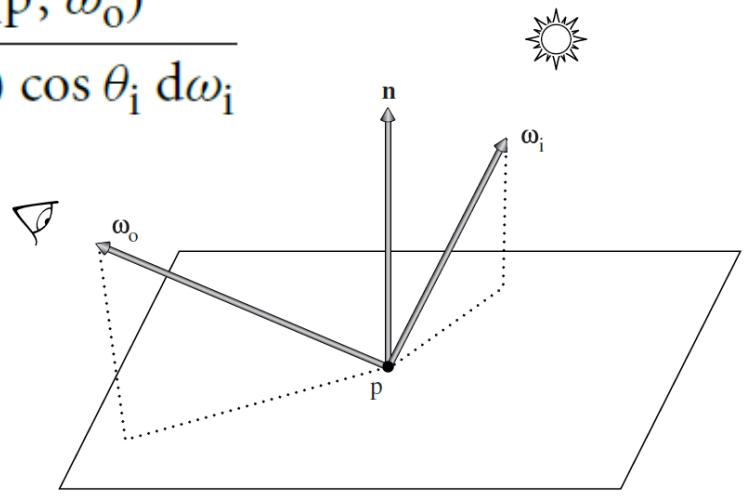
- **Basic idea**
  - What if we were to render the scene from the light's perspective and store the resulting depth values in a texture?



Bidirectional reflectance distribution function

- **Bidirectional reflectance distribution function (BRDF)**
  - A formalism for describing reflection from a surface
  - How much radiance is leaving the surface as a result of incident radiance

$$f_r(\mathrm{p}, \omega_o, \omega_i) = \frac{\mathrm{d}L_o(\mathrm{p}, \omega_o)}{\mathrm{d}E(\mathrm{p}, \omega_i)} = \frac{\mathrm{d}L_o(\mathrm{p}, \omega_o)}{L_i(\mathrm{p}, \omega_i)\cos\theta_i \,\mathrm{d}\omega_i}$$



Rendering equation

- **The fundamental rendering equation**
  - Describe how an incident distribution of light at a point is transformed into an outgoing distribution

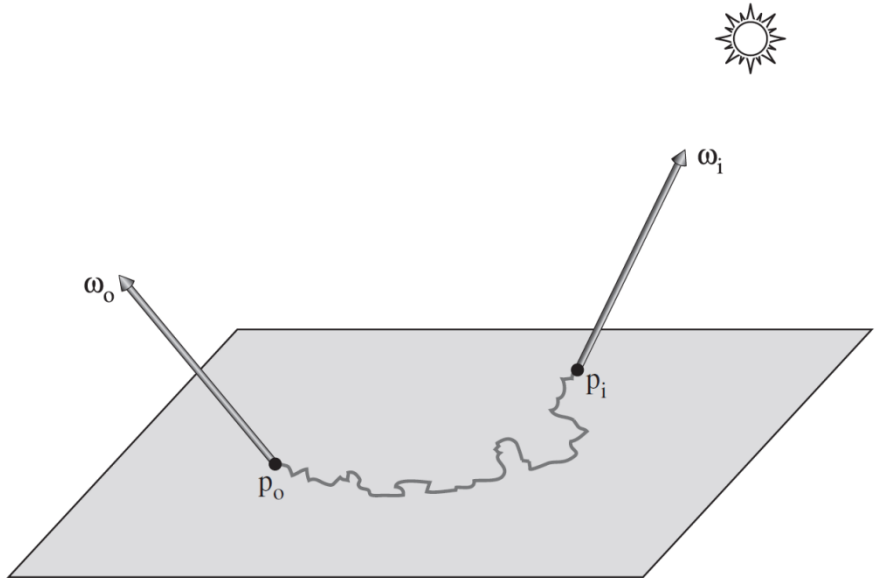
$$L_o(p, \omega_o) = \int_{S^2} f(p, \omega_o, \omega_i)\, L_i(p, \omega_i)\, |\cos \theta_i|\, d\omega_i$$

  - When S^2 (the entire sphere) is the domain, it is often called the *scattering equation*
  - When upper hemisphere is the domain, it is often called the *reflection equation*

Bidirectional scattering-surface reflectance distribution function

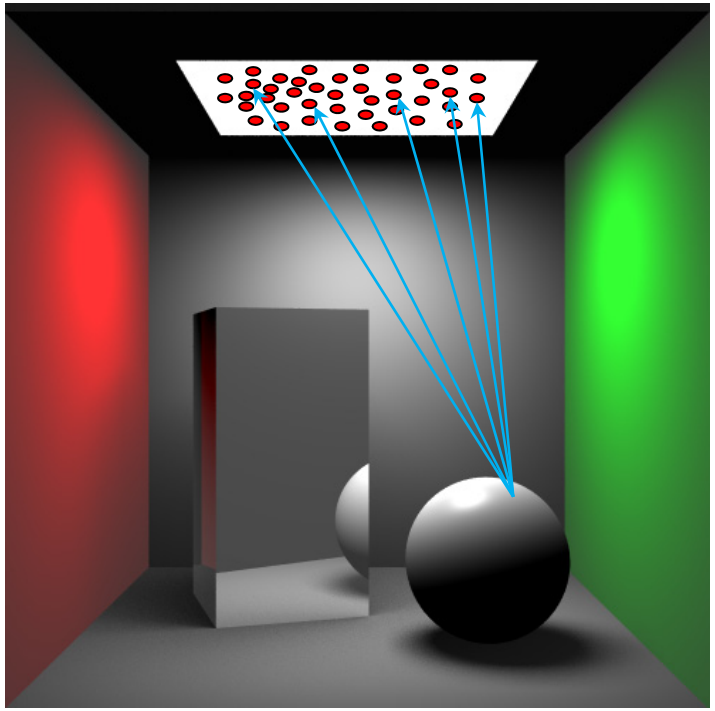
- **Generalization of scattering equation**
  - Turn fundamental rendering equation into more complex integral equation

$$L_o(p_o, \omega_o) = \int_A \int_{\mathcal{H}^2(n)} S(p_o, \omega_o, p_i, \omega_i)\, L_i(p_i, \omega_i)\, |\cos \theta_i|\, d\omega_i\, dA$$

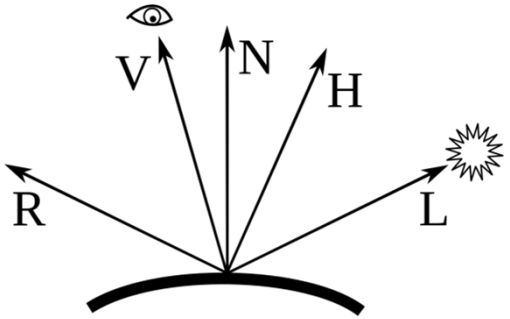


More on lighting

- **Breaking the limit of point light source**
  - Sample on area light source
  - Sum together all the contributions from point light sources

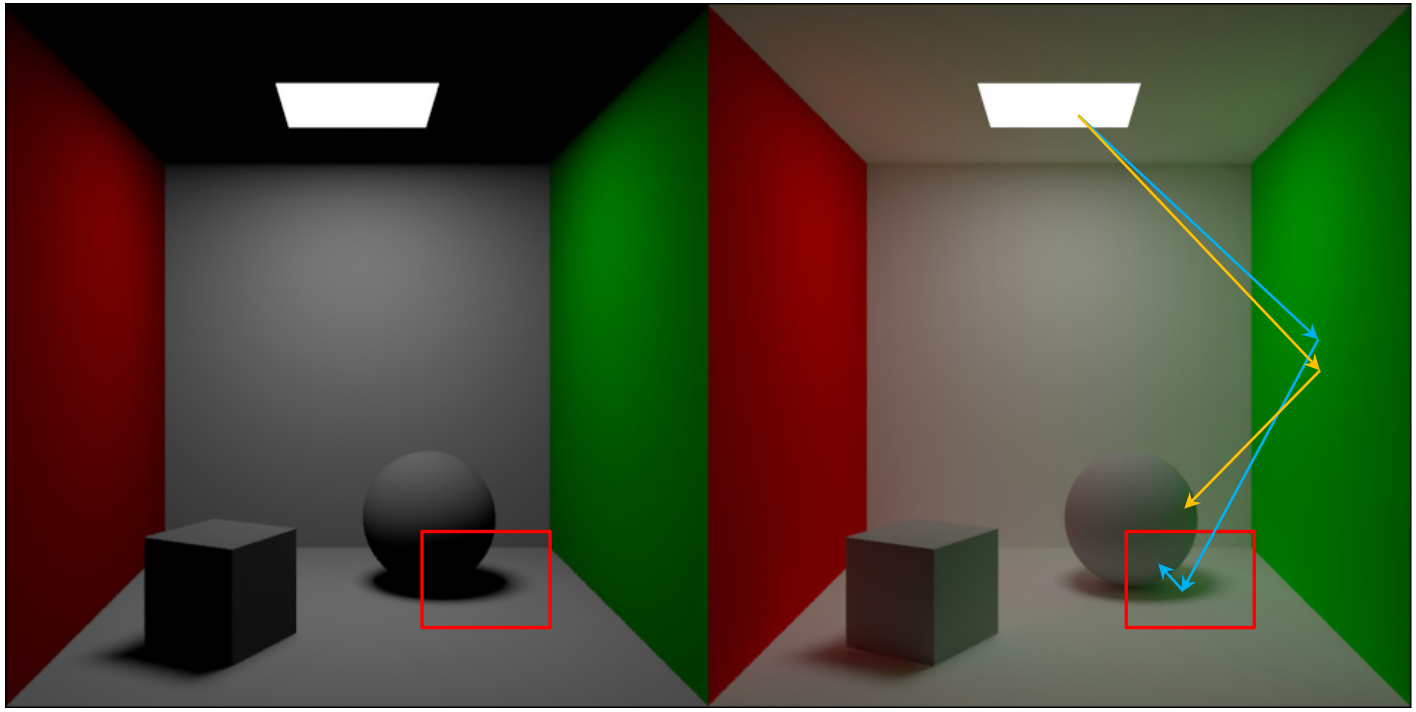


$$I_{\mathrm{p}} = k_{\mathrm{a}} i_{\mathrm{a}} + \sum_{m \in \text{lights}} (k_{\mathrm{d}} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_{\mathrm{s}} (\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,\mathrm{s}})$$



Illumination in a scene

- **Global illumination**
  - Illumination cast on objects from both light sources and surface inter-reflections
  - Direct illumination + indirect illumination



Light transport equation

- **Partitioning the integrand**
  - We can decompose the path integral into three components

$$L(\mathrm{p}_1 \rightarrow \mathrm{p}_0) = P(\bar{\mathrm{p}}_1) + P(\bar{\mathrm{p}}_2) + \sum_{i=3}^{\infty} P(\bar{\mathrm{p}}_i)$$

    - First term
      - Emitted radiance at p_1
    - Second term
      - Solved with an accurate direct lighting solution
    - Third term (indirect lighting)
      - Solved with faster but less accurate approach

Ray sampling

- **Inversion method**
  - Application to continuous random variables
    - 1. Compute the CDF
    - 2. Compute the inverse
    - 3. Obtain a uniformly distributed random number ξ
    - 4. Compute

Ray sampling

- **Metropolis sampling pseudo-code**

```
X = X0
for i = 1 to n
    X' = mutate(X)
    a = accept(X, X')
    if (random() < a)
        X = X'
    record(X)
```

  - The recorded *X* sequence will be used for integration

Transforming between distributions

- **Function of a random variable**
  - Suppose we are given random variable X_i with PDF $p_x(x)$
  - Given $Y_i=y(X_i)$, the following equality satisfies

$$Pr\{Y \le y(x)\} = Pr\{X \le x\} \quad \Rightarrow \quad P_y(y) = P_y(y(x)) = P_x(x)$$

  - Differentiating

$$p_y(y)\frac{\mathrm{d}y}{\mathrm{d}x} = p_x(x) \quad \Rightarrow \quad p_y(y) = \left(\frac{\mathrm{d}y}{\mathrm{d}x}\right)^{-1} p_x(x)$$

  - Usually we know $p_y(y)$ (and P(y)), how to sample y?

$$y(x) = P_y^{-1}\left(P_x(x)\right)$$

Transforming between distributions

- **Transformation in multiple dimensions**
  - Let n-dimensional random variable X with density function $p_x(x)$
  - Let Y=T(X), T is a bijective mapping

$$p_y(y) = p_y(T(x)) = \frac{p_x(x)}{|J_T(x)|}$$

- Jacobian:

$$\begin{pmatrix} \partial T_1/\partial x_1 & \cdots & \partial T_1/\partial x_n \\ \vdots & \ddots & \vdots \\ \partial T_n/\partial x_1 & \cdots & \partial T_n/\partial x_n \end{pmatrix}$$

Transforming between distributions

- **Example**
  - Polar coordinates

$$x = r\cos\theta$$
$$y = r\sin\theta$$

  - Suppose we draw samples from some density p(r,θ)
  - Computing the Jacobian

$$J_T = \begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{pmatrix}$$

  - The determinant: $r\left(\cos^2\theta + \sin^2\theta\right) = r$
  - So

$$p(x, y) = p(r, \theta)/r \quad \Rightarrow \quad p(r, \theta) = r\, p(x, y)$$

Transforming between distributions

- **Example**
  - Spherical coordinates

$$x = r \sin\theta \cos\phi$$
$$y = r \sin\theta \sin\phi$$
$$z = r \cos\theta$$

- Computing the Jacobian determinant: $|J_T| = r^2 \sin\theta$
- The corresponding density function

$$p(r, \theta, \phi) = r^2 \sin\theta \; p(x, y, z)$$

- Solid angle defined with spherical coordinates $d\omega = \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi$
- If we have a density function defined over a solid angle

$$p(\theta, \phi) \, \mathrm{d}\theta \, \mathrm{d}\phi = p(\omega) \, \mathrm{d}\omega \quad \Rightarrow \quad p(\theta, \phi) = \sin\theta \; p(\omega)$$

2D sampling

- **Example**
  - Sampling a unit disk uniformly
    - Wrong approach: $r = \xi_1, \theta = 2\pi\xi_2$
    - PDF p(x,y) by normalization is: $p(x, y) = 1/\pi$
    - Transform into polar coordinate: $p(r, \theta) = r/\pi$ $\quad p(r, \theta) = r\ p(x, y)$
    - Compute the marginal and conditional densities

$$p(r) = \int_0^{2\pi} p(r, \theta)\, \mathrm{d}\theta = 2r$$

$$p(\theta|r) = \frac{p(r, \theta)}{p(r)} = \frac{1}{2\pi}$$

    - Integrating and inverting to find P(r), $P^{-1}(r)$, P(θ), and $P^{-1}(\theta)$

$$r = \sqrt{\xi_1}$$

$$\theta = 2\pi\xi_2$$

3D sampling

- **Example**
  - Uniformly sampling a hemisphere
    - Uniform sampling means $p(\omega) = c$
    - Normalization:

$$\int_{\mathcal{H}^2} p(\omega)\, \mathrm{d}\omega = 1 \Rightarrow c \int_{\mathcal{H}^2} \mathrm{d}\omega = 1 \Rightarrow c = \frac{1}{2\pi} \longrightarrow p(\omega) = 1/(2\pi) \longrightarrow p(\theta, \phi) = \sin\theta/(2\pi)$$

$$p(\theta, \phi) = \sin\theta\ p(\omega)$$

    - Consider sampling θ:

$$p(\theta) = \int_0^{2\pi} p(\theta, \phi)\, \mathrm{d}\phi = \int_0^{2\pi} \frac{\sin\theta}{2\pi}\, \mathrm{d}\phi = \sin\theta$$

    - Compute the conditional density for φ:

$$p(\phi|\theta) = \frac{p(\theta, \phi)}{p(\theta)} = \frac{1}{2\pi}$$

3D sampling

- **Example**
  - Uniformly sampling a hemisphere
    - Use 1D inversion technique to sample:

$$P(\theta) = \int_0^\theta \sin\theta' \, d\theta' = 1 - \cos\theta$$

$$P(\phi|\theta) = \int_0^\phi \frac{1}{2\pi} \, d\phi' = \frac{\phi}{2\pi}$$

    - Inversion is straightforward

$$\theta = \cos^{-1}\xi_1$$
$$\phi = 2\pi\xi_2.$$

$$x = \sin\theta\cos\phi = \cos(2\pi\xi_2)\sqrt{1-\xi_1^2}$$
$$y = \sin\theta\sin\phi = \sin(2\pi\xi_2)\sqrt{1-\xi_1^2}$$
$$z = \cos\theta = \xi_1$$

Importance sampling

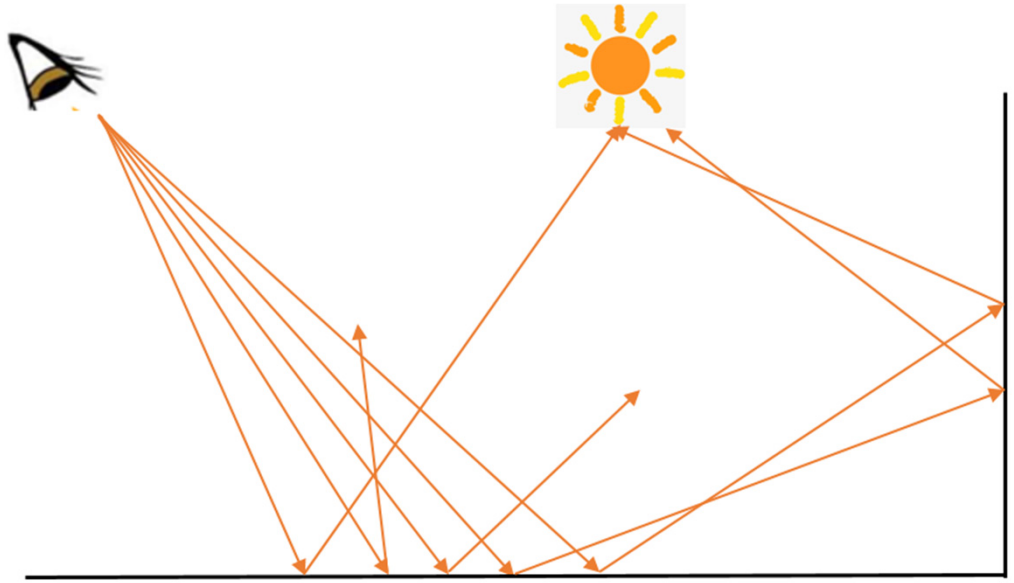
- **Importance sampling is a variance reduction technique**
  - Monte-Carlo estimator

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

  - The fact
    - If samples are taken from distribution p(x) that is similar to function f(x), the convergence will be much faster
    - Can increase variance is p(x) is bad
  - Basic idea
    - Concentrate work where the values of the integrand is relatively high

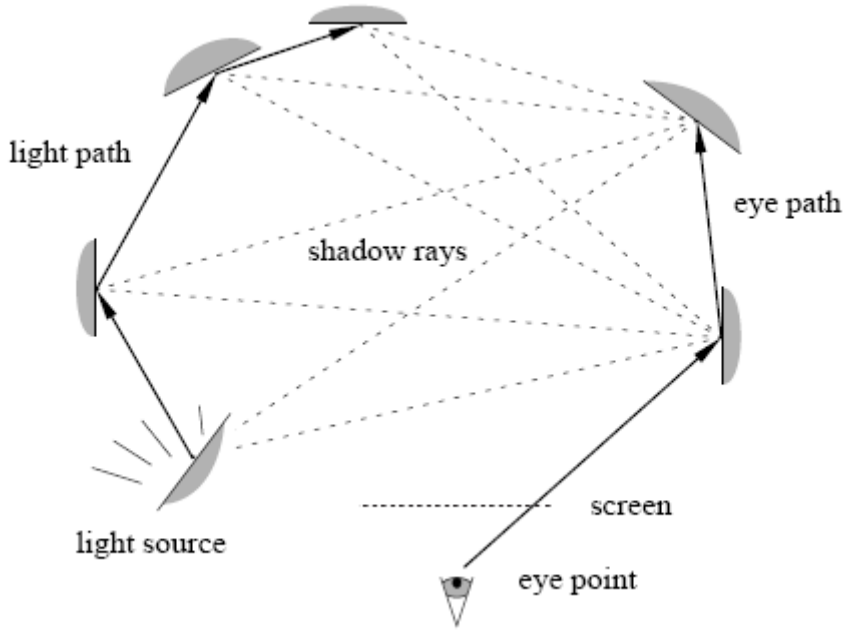
Path tracing

- **Approximation**
  - Instead of shooting multiple rays per intersection, we shoot only one ray
  - Instead of shooting only a few rays per pixel, we should large amount of rays per pixel



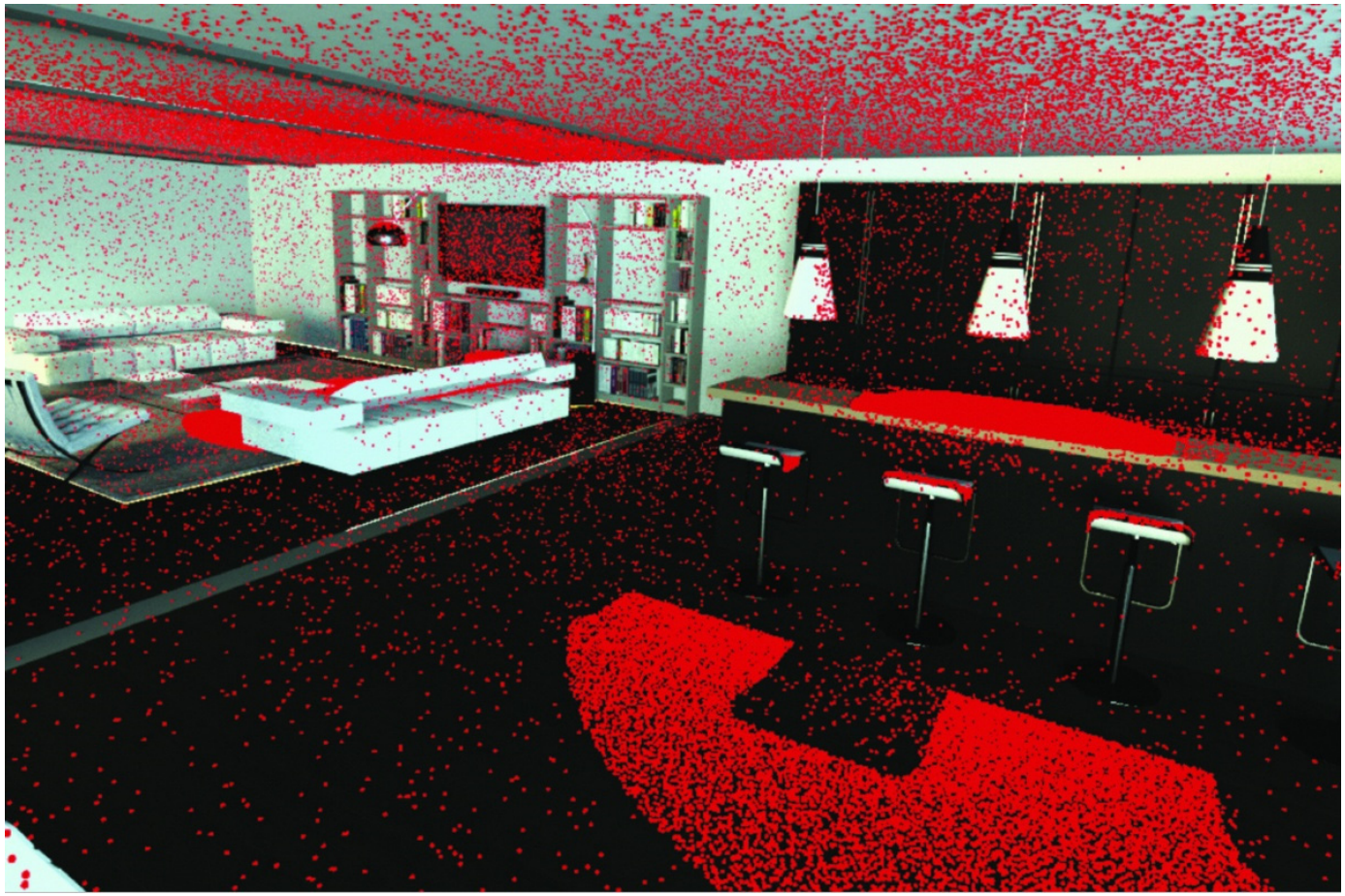
Bidirectional path tracing

- **For each vertex on camera path**
  - Check visibility for each vertex on light path
  - Render each sampled path with BSDF



Photon mapping

- **Photon distribution**



Photon mapping

- **Radiance estimate**
  - Incoming flux is approximated using the photon map
  - Searching the nearest n photons
  - Each photon p has equal power (energy)

$$L_r(x,\vec{\omega}) = \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) \frac{d\Phi_i(x,\vec{\omega}')}{dA_i} \approx \sum_{p=1}^{n} f_r(x,\vec{\omega}_p,\vec{\omega}) \frac{\Delta\Phi_p(x,\vec{\omega}_p)}{\Delta A}$$

  - Assuming that the surface is locally flat
    - Projecting the sphere onto the tangent surface

$$\Delta A = \pi r^2$$

7. Texturing and Image Processing

What is a texture?

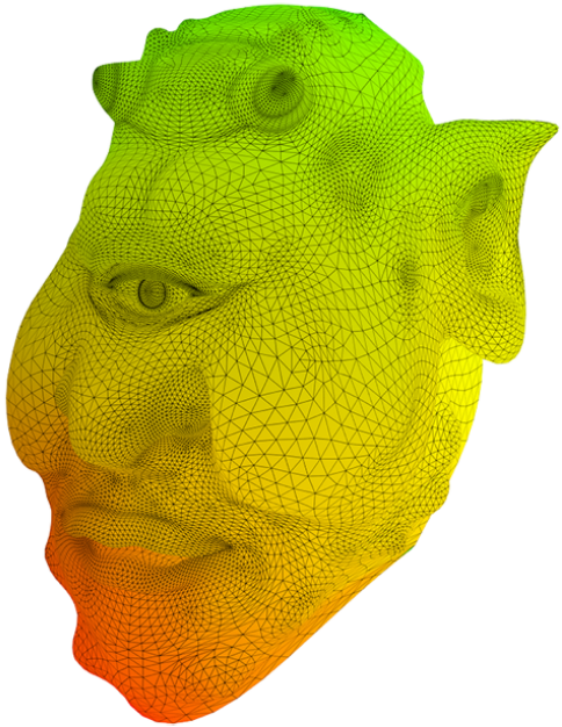
- **How can we add surface details?**



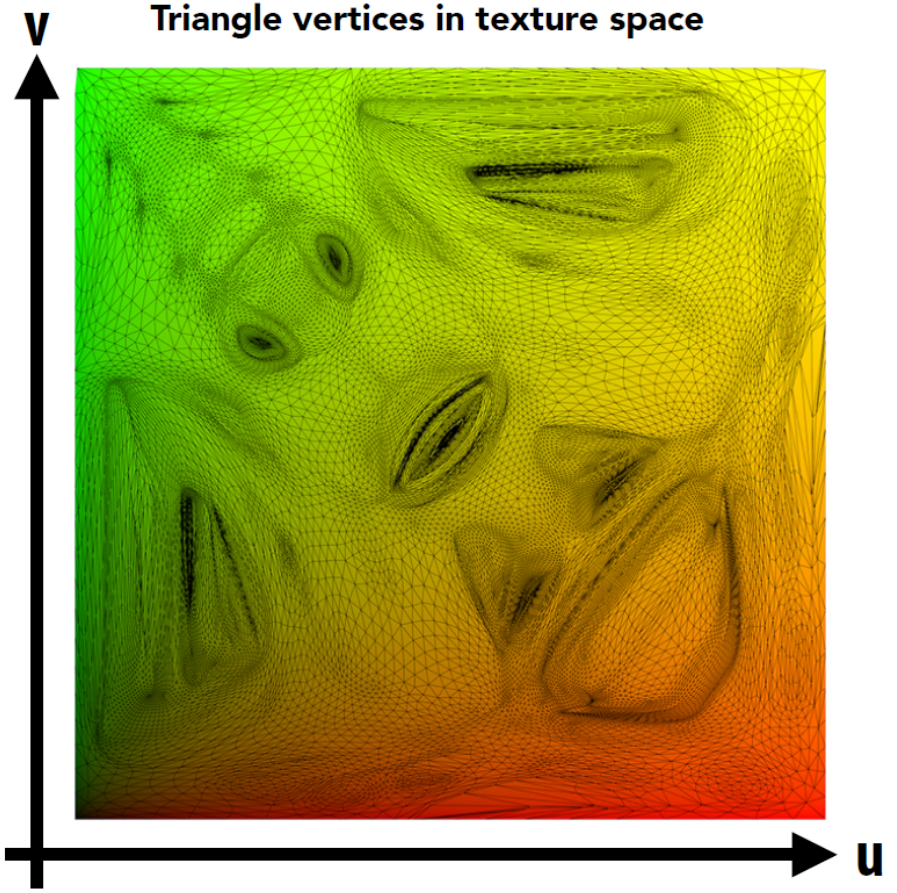

- **Definition**
  - Narrow: surface variations
  - Broad: an image representing surface details

Visualization of texture coordinates

Visualization of texture coordinates

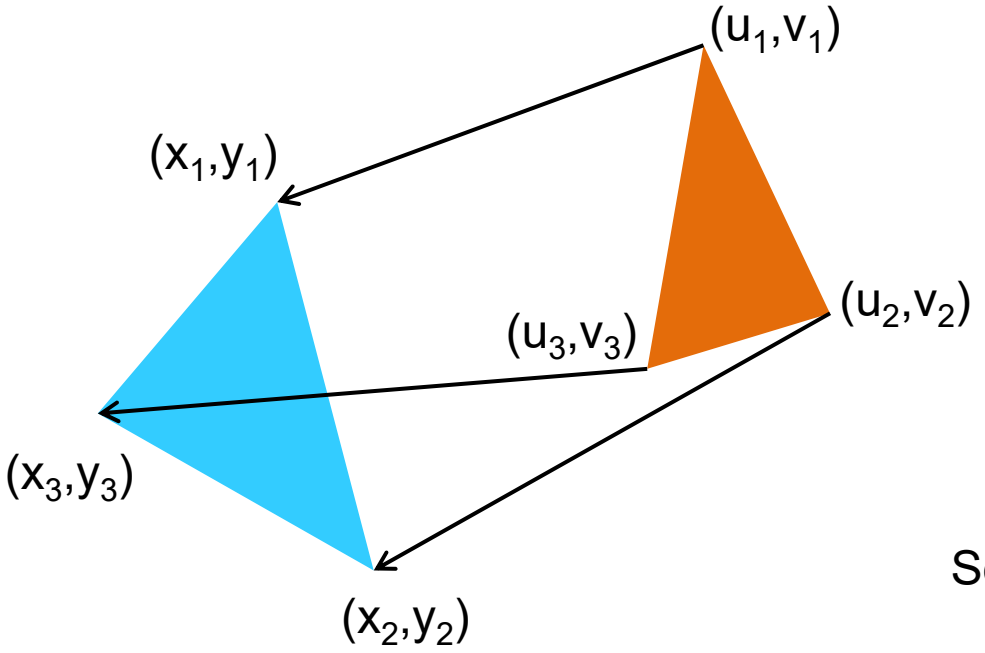


Triangle vertices in texture space



2D texture mapping

- **How to perform texture mapping?**
  - Construct the texture projection matrix
  - Mapping between projected vertices and texture coordinates



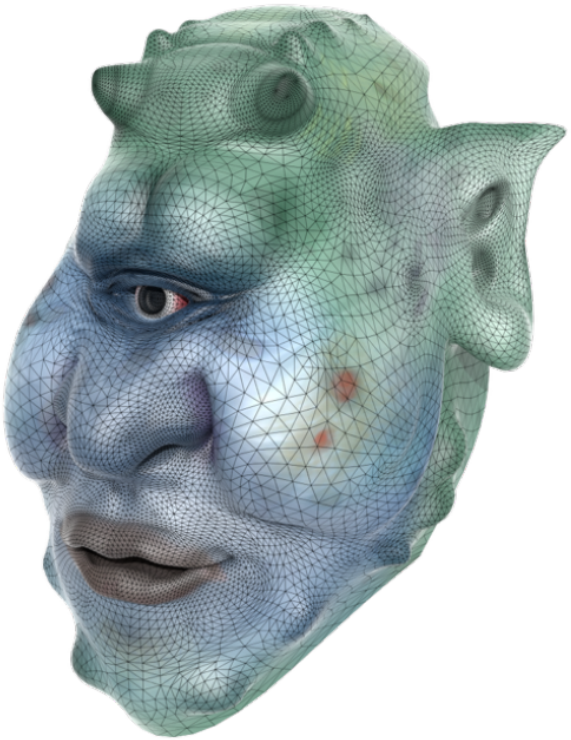
$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$$

0 0 1

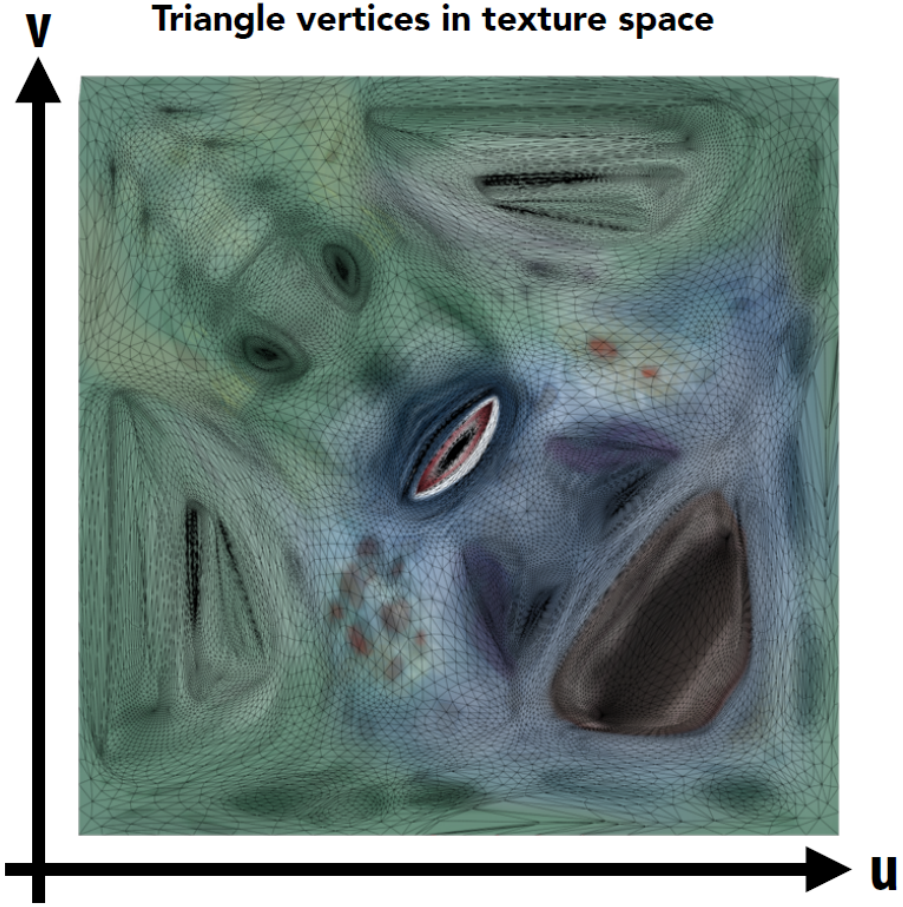
Solve a linear system for a triangle pair

Image texture applied to a surface

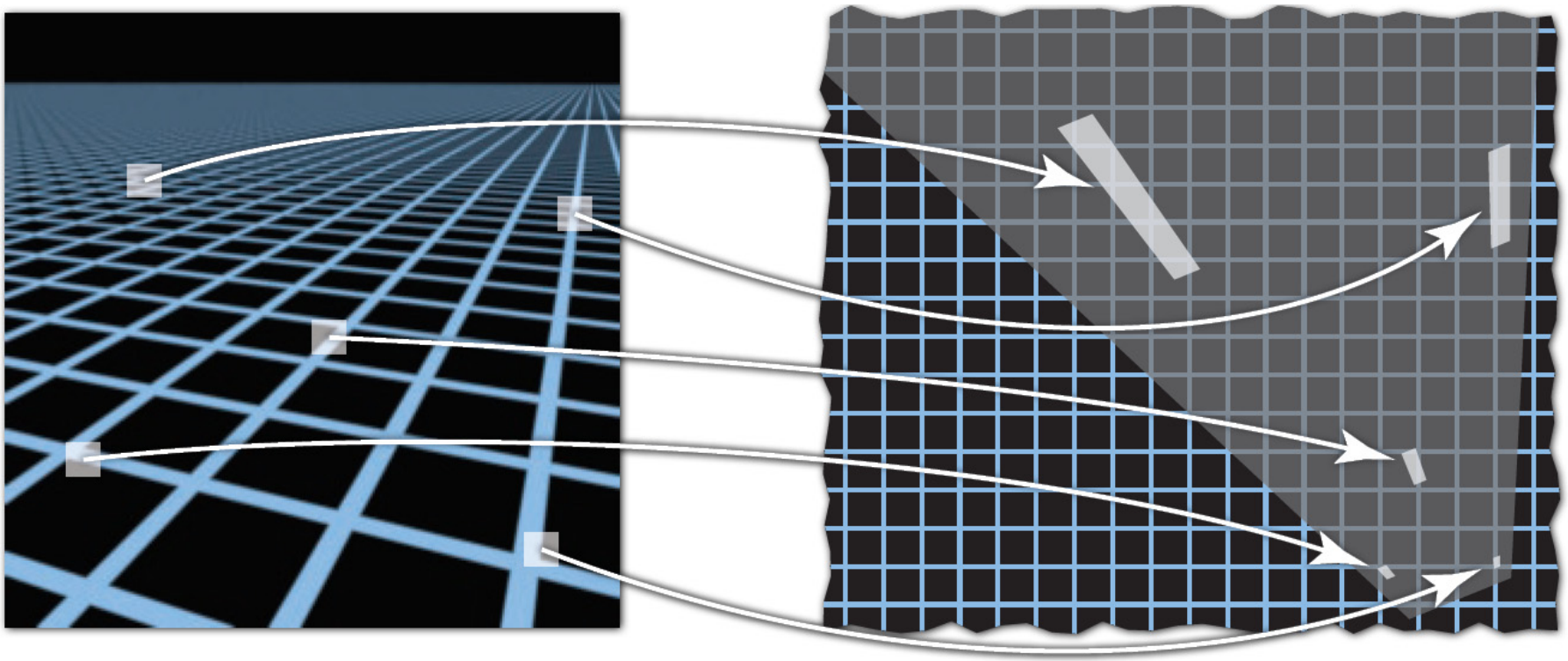
Rendered result



Triangle vertices in texture space



Screen pixel footprint in texture

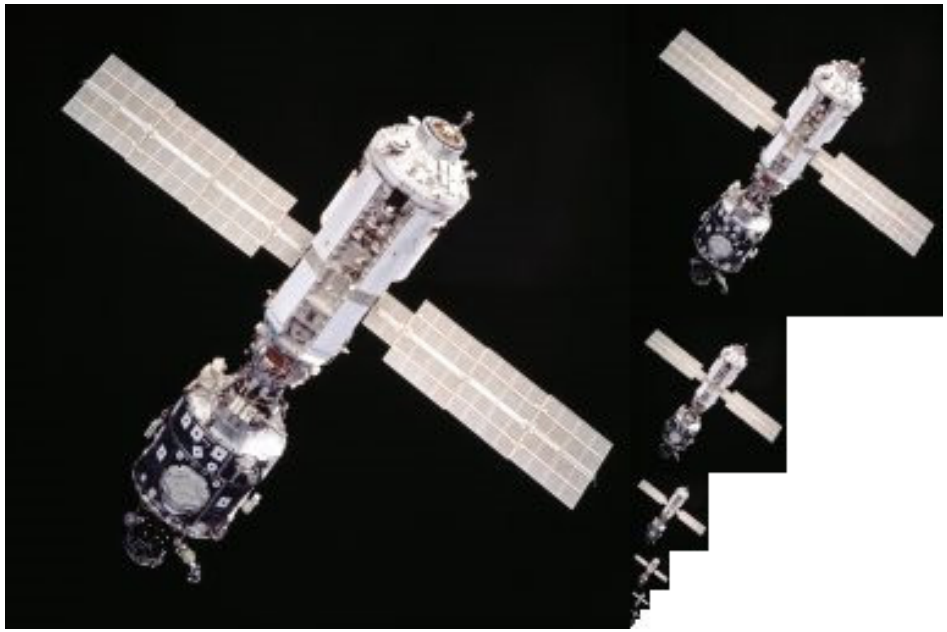


Screen space

Texture space

Mipmap

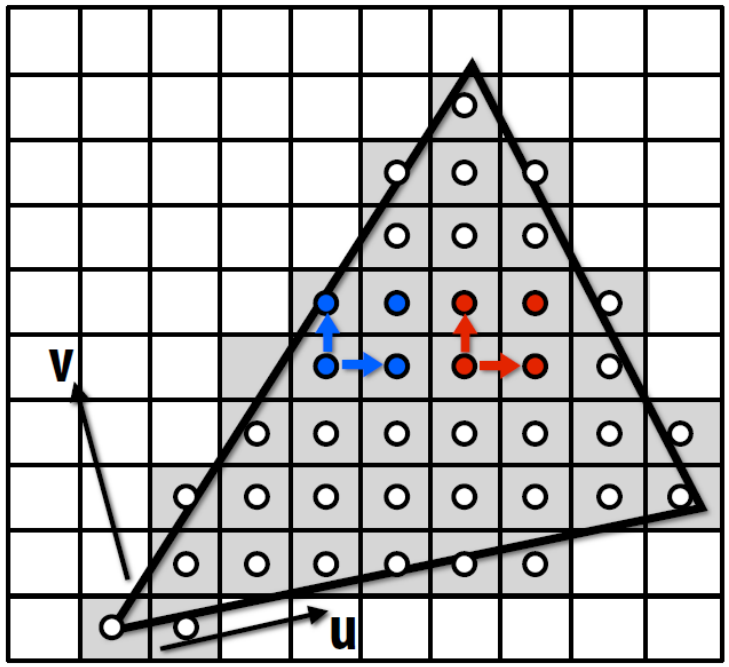
- **Mipmap (L. Williams 1983)**



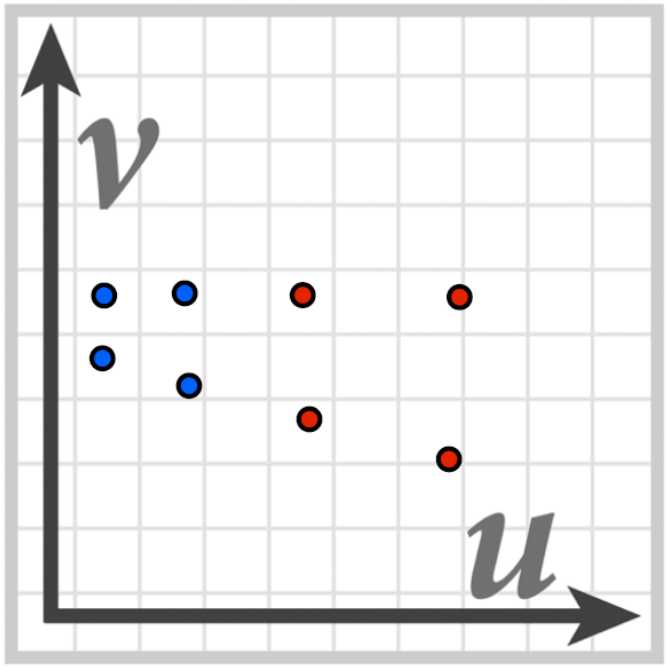
“Mip” comes from the Latin “multum in parvo", meaning a multitude in a small space

Mipmap

- **Computing mipmap level D**
  - Estimate texture footprint using texture coordinates of neighboring screen samples



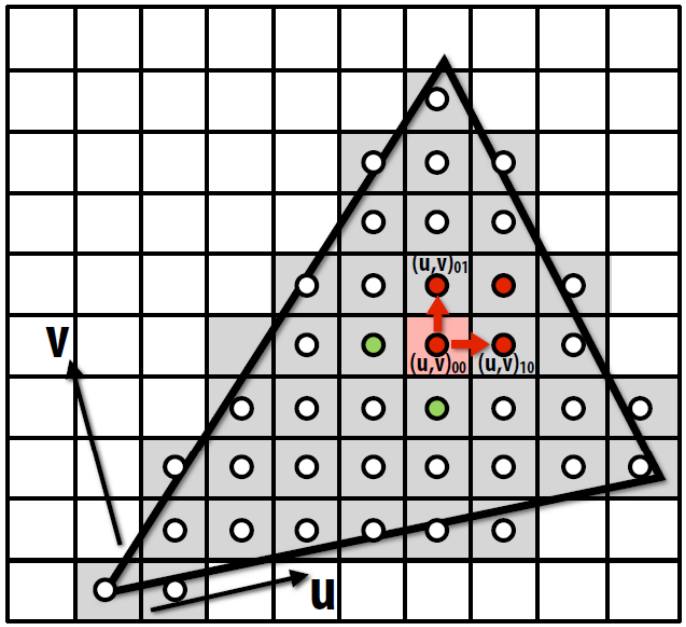
Screen space

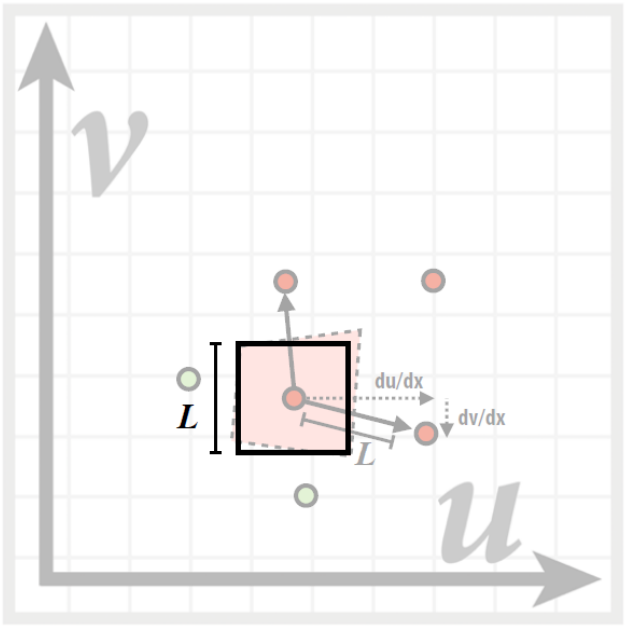


Texture space

Mipmap

- **Computing mipmap level D**



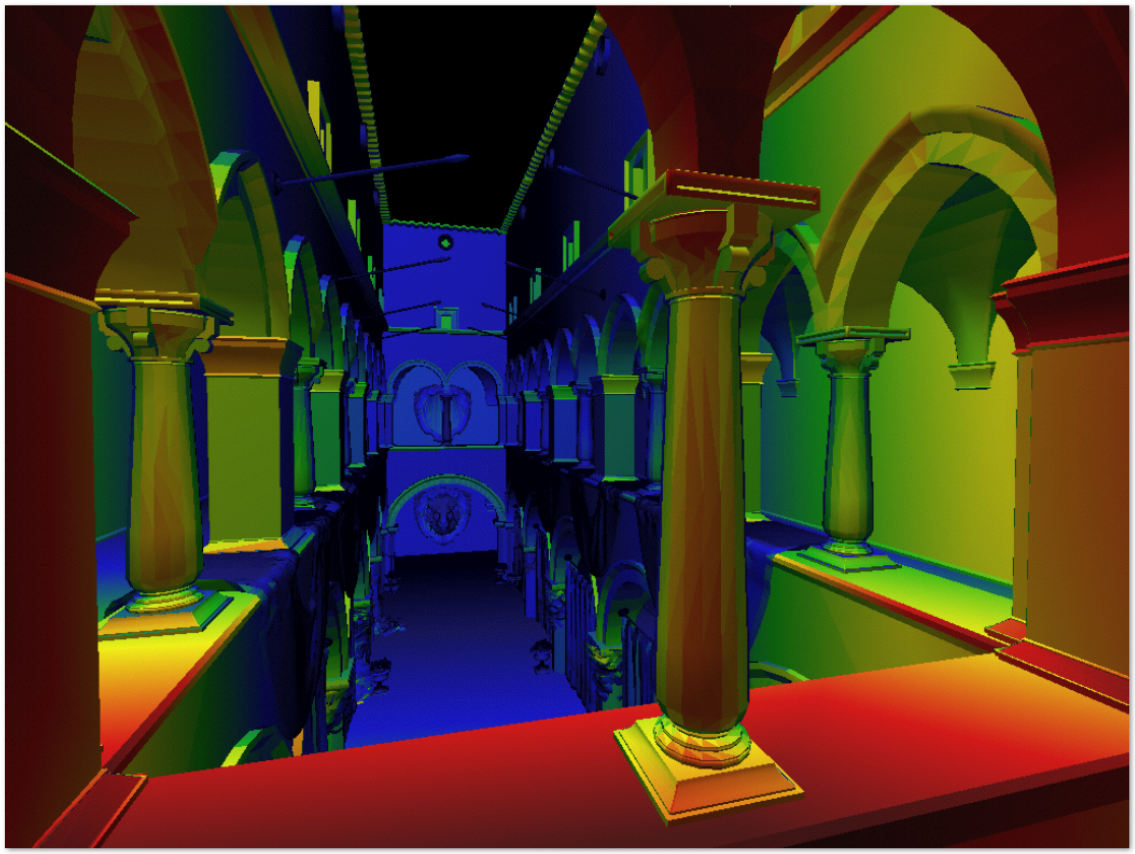


$$D = \log_2 L$$

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

Mipmap

- **Visualization of mipmap level**



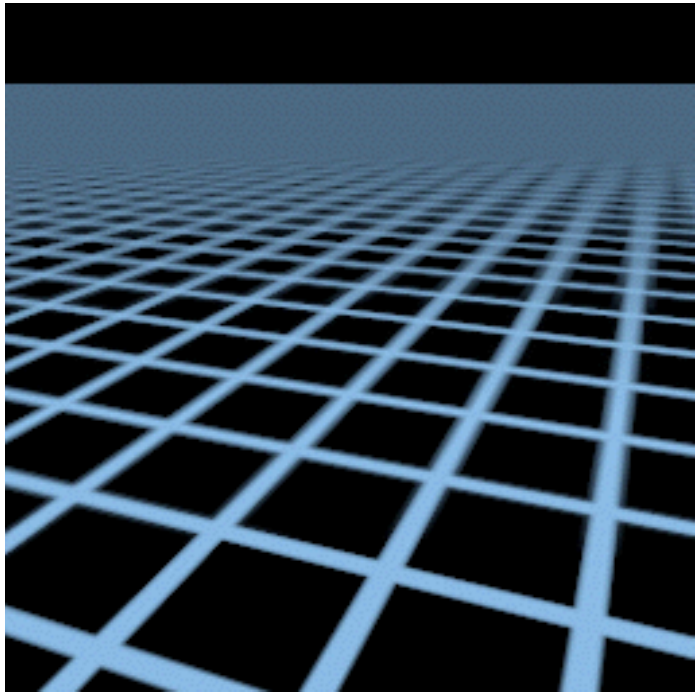
Trilinear filtering: visualization of continuous D

Anisotropic filtering

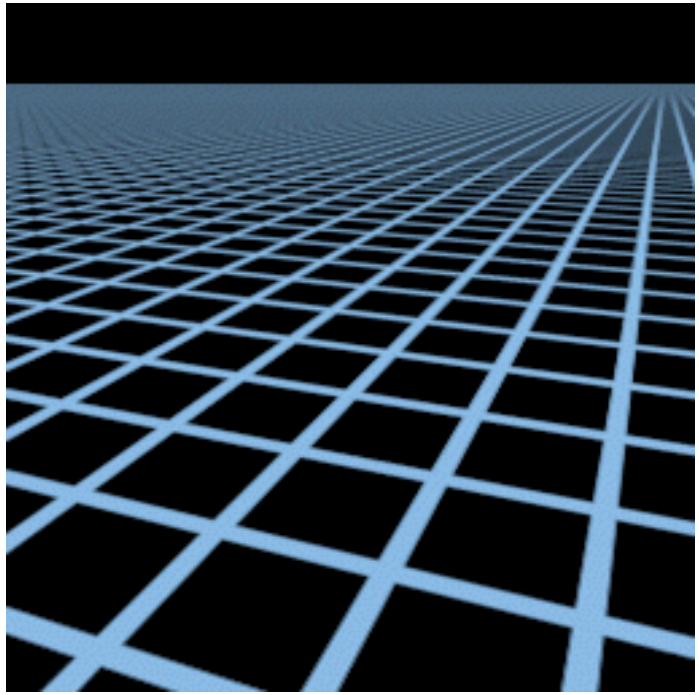
- **Ripmap**

Anisotropic filtering

- **Elliptical weighted average filtering**



Mipmap



Ripmap

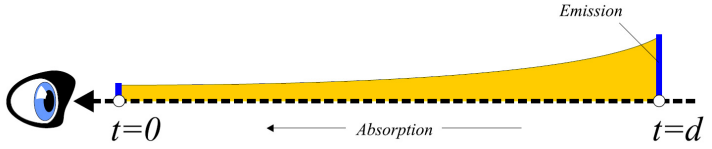
8. Volume Rendering

Overview of volume rendering

- **Volume rendering equation**
  - Integrating along the direction of light

$$I(D) = I_0\, e^{-\int_{s_0}^{D} \kappa(t)\,\mathrm{d}t} + \int_{s_0}^{D} q(s)\, e^{-\int_{s}^{D} \kappa(t)\,\mathrm{d}t}\,\mathrm{d}s$$

  - Integral in terms of transparency



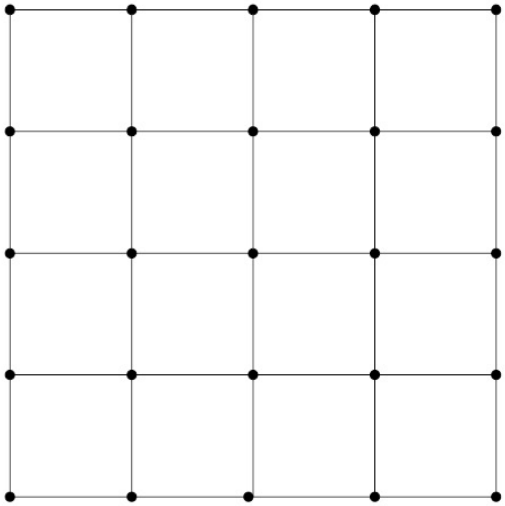
$$I(D) = I_0\, T(s_0, D) + \int_{s_0}^{D} q(s) T(s, D)\,\mathrm{d}s$$

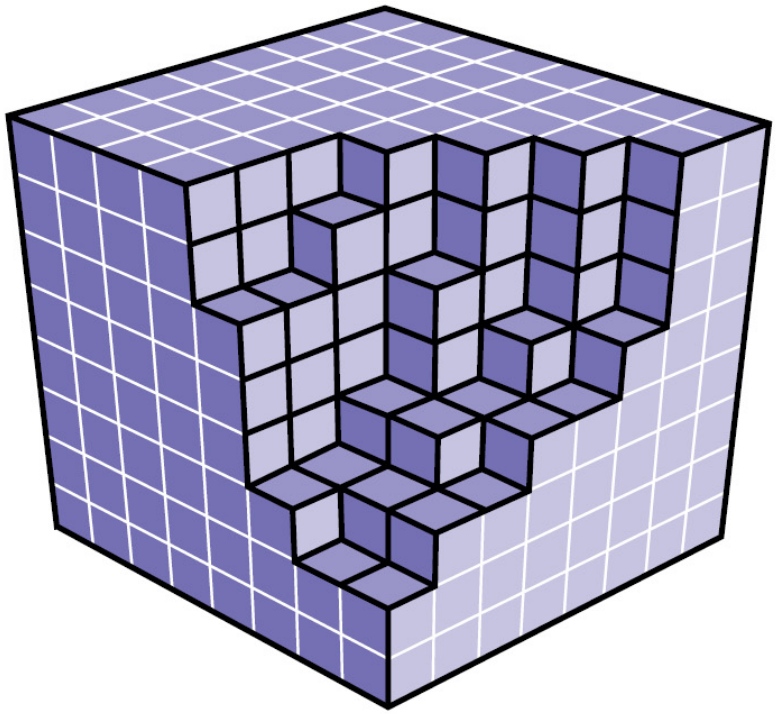
Reduction of energy

Production of energy

Overview of volume rendering

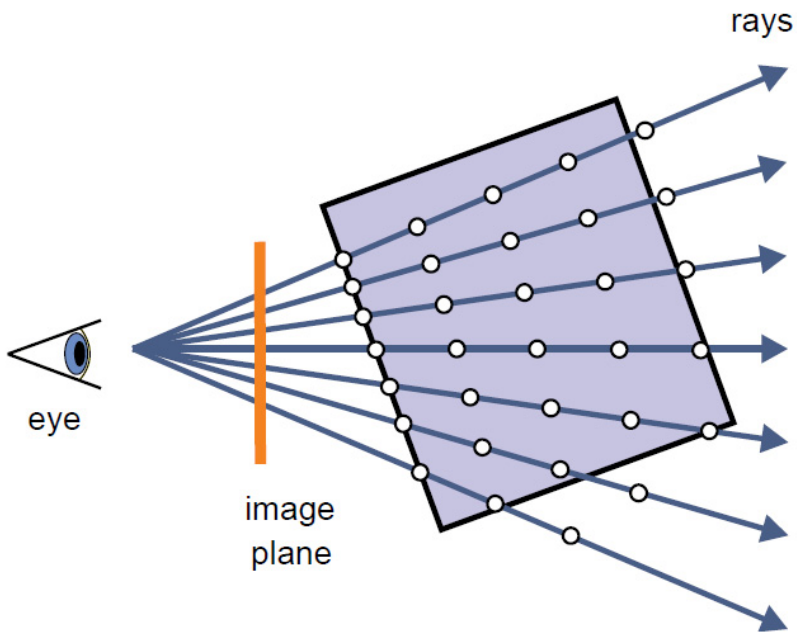
- **Storage of volume data**
  - **Uniform grid**
    - From pixel to voxel
    - Sample a volumetric space with voxels





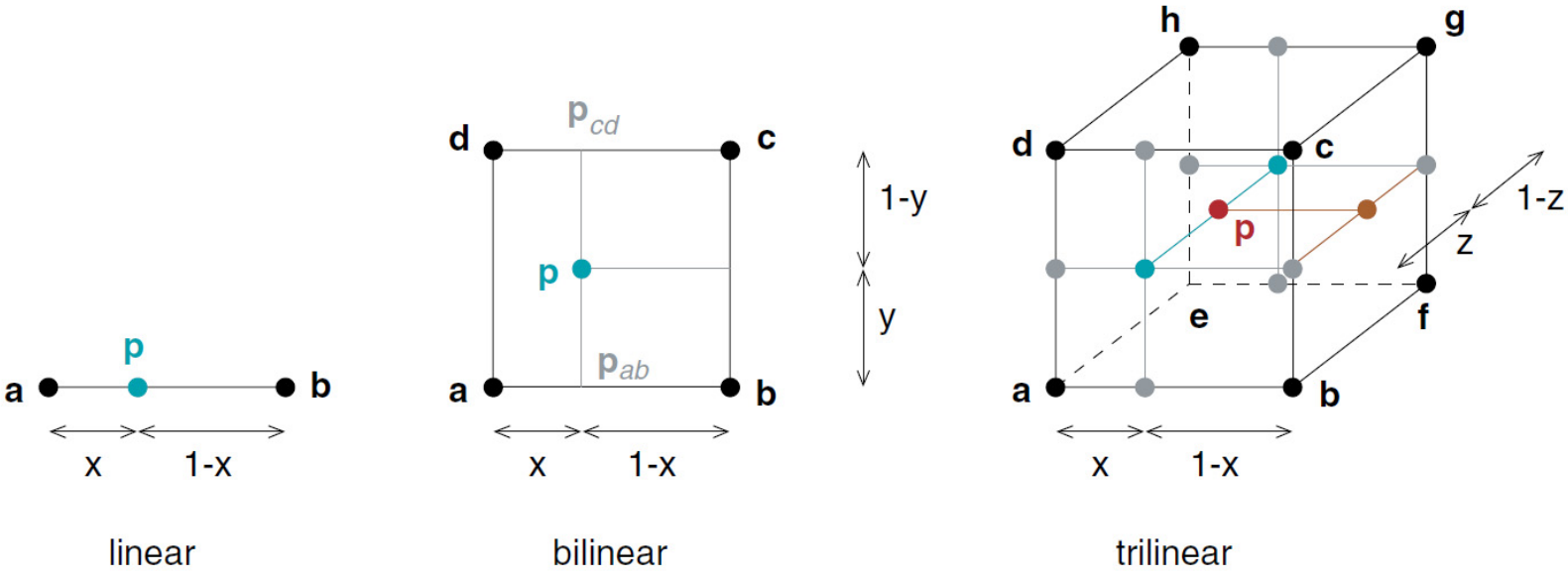
Overview of volume rendering

- **Ray casting**
  - The most popular image-order method for volume rendering
  - Directly evaluate volume-rendering integral along rays



Overview of volume rendering

- **Sampling & reconstruction**
  - Linear interpolation
  - Cubic-spline interpolation

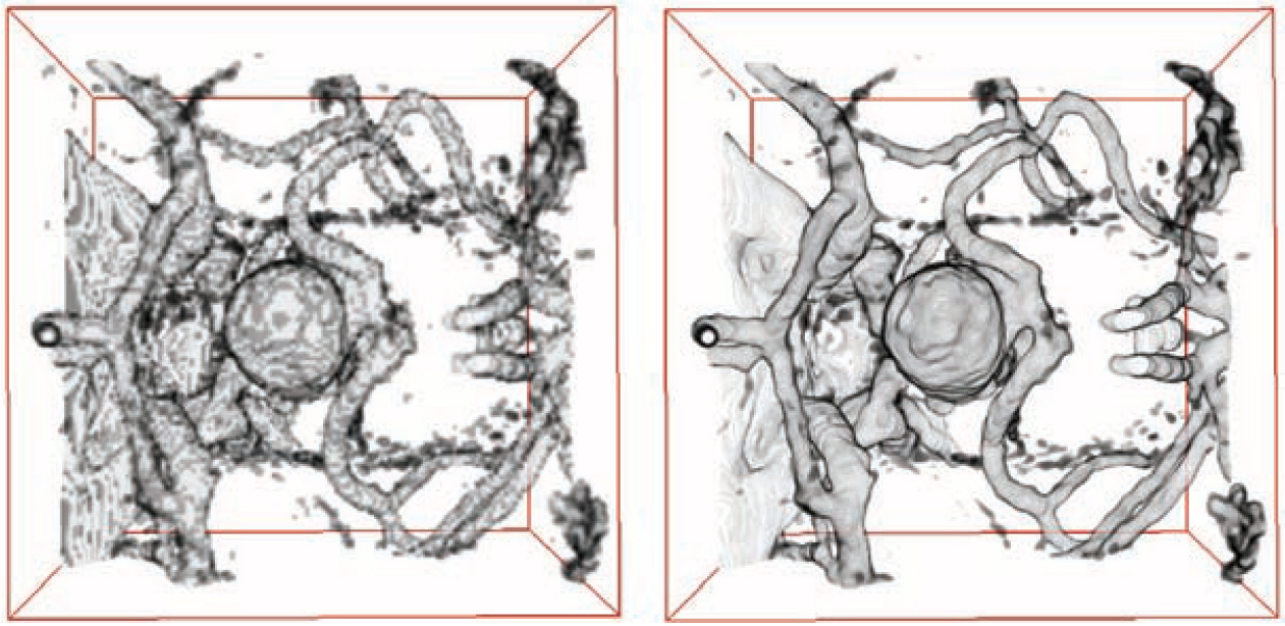


Overview of volume rendering

- **Transfer function**
  - **Optical properties for volume data**
    - Emission, absorption, scattering
  - **Given any scalar volume data**
    - Determine optical properties
    - Reflect certain property when rendered
    - E.g., surface represented by volume data (volume modeling)
    - Transfer function: a mapping from arbitrary scalar volume data to optical properties
  - **May contain high frequencies**
    - Classification of data

Overview of volume rendering

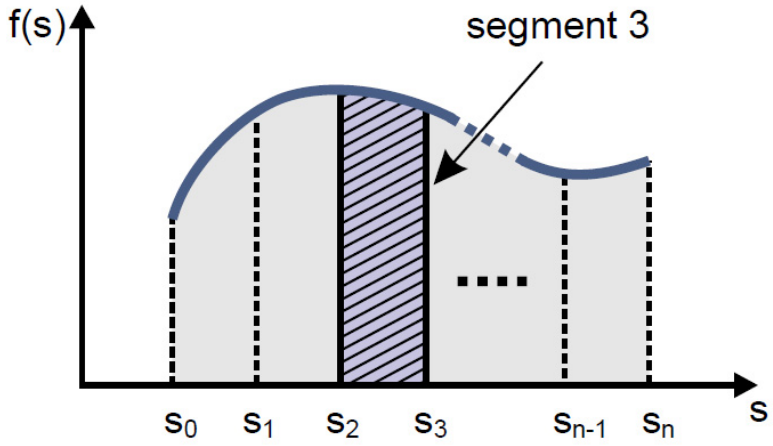
- **Pre/post-interpolative transfer functions**
  - **Pre-interpolative transfer function**
    - Apply a transfer function before data interpolation
  - **Post-interpolative transfer function**
    - Apply a transfer function after data interpolation



Overview of volume rendering

- **Ray integral**
  - Split the integration domain into n subsequent intervals: $s_o < s_1 < \cdots < s_{n-1} < s_n$
  - Consider light transport within the i-th interval $[s_{i-1}, s_i]$

$$I(s_i) = I(s_{i-1})T(s_{i-1}, s_i) + \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)\,\mathrm{d}s$$



Discretization

- **Ray integral**
  - Quantity definition

$$I(s_i) = I(s_{i-1})T(s_{i-1}, s_i) + \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)\,\mathrm{d}s$$

$$T_i = T(s_{i-1}, s_i)\,, \quad c_i = \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)\,\mathrm{d}s$$

$$I(D) = I(s_n) = I(s_{n-1})T_n + c_n = \left(I(s_{n-2})T_{n-1} + c_{n-1}\right)T_n + c_n = \dots$$

$$I(D) = \sum_{i=0}^{n} c_i \prod_{j=i+1}^{n} T_j\,, \quad \text{with } c_0 = I(s_0)$$

Overview of volume rendering

- **Compositing schemes**
  - **Front-to-back composition**
    - Viewing rays are traversed from eye into the volume

$$\hat{C}_i = \hat{C}_{i+1} + \hat{T}_{i+1} C_i$$
$$\hat{T}_i = \hat{T}_{i+1}(1 - \alpha_i)$$

Alpha blending →

$$C_{\text{dst}} \leftarrow C_{\text{dst}} + (1 - \alpha_{\text{dst}}) C_{\text{src}}$$
$$\alpha_{\text{dst}} \leftarrow \alpha_{\text{dst}} + (1 - \alpha_{\text{dst}}) \alpha_{\text{src}}$$

  - **Back-to-front composition**
    - Viewing rays are traversed from back of the volume into the eye

$$\hat{C}_i = \hat{C}_{i-1}(1 - \alpha_i) + C_i$$
$$\hat{T}_i = \hat{T}_{i-1}(1 - \alpha_i)$$

Alpha blending →

$$C_{\text{dst}} \leftarrow (1 - \alpha_{\text{src}}) C_{\text{dst}} + C_{\text{src}}$$

Next Lecture : Computer animation - nonphysically-based techniques