CS354 Computer Graphics
Midterm Review

Qixing Huang

March 28th 2018

Elements of Rendering

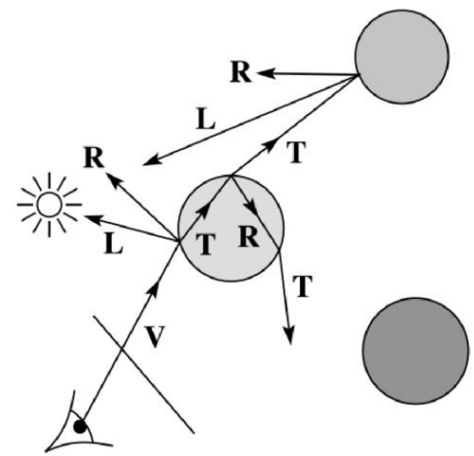
- Object
- Light
- Material
- Camera

Geometric optics

- Modern theories of light treat it as both a wave and a particle
- We will take a combined and somewhat simpler view of light – the view of geometric optics
- Here are the rules of geometric optics:
  - Light is a flow of photons with wavelengths. We'll call these flows "light rays"
  - Light rays travel in straight lines in free space
  - Light rays do not interfere with each other as they cross
  - Light rays obey the laws of reflection and refraction
  - Light rays travel from the light sources to the eye, but the physics is invariant under path reversal

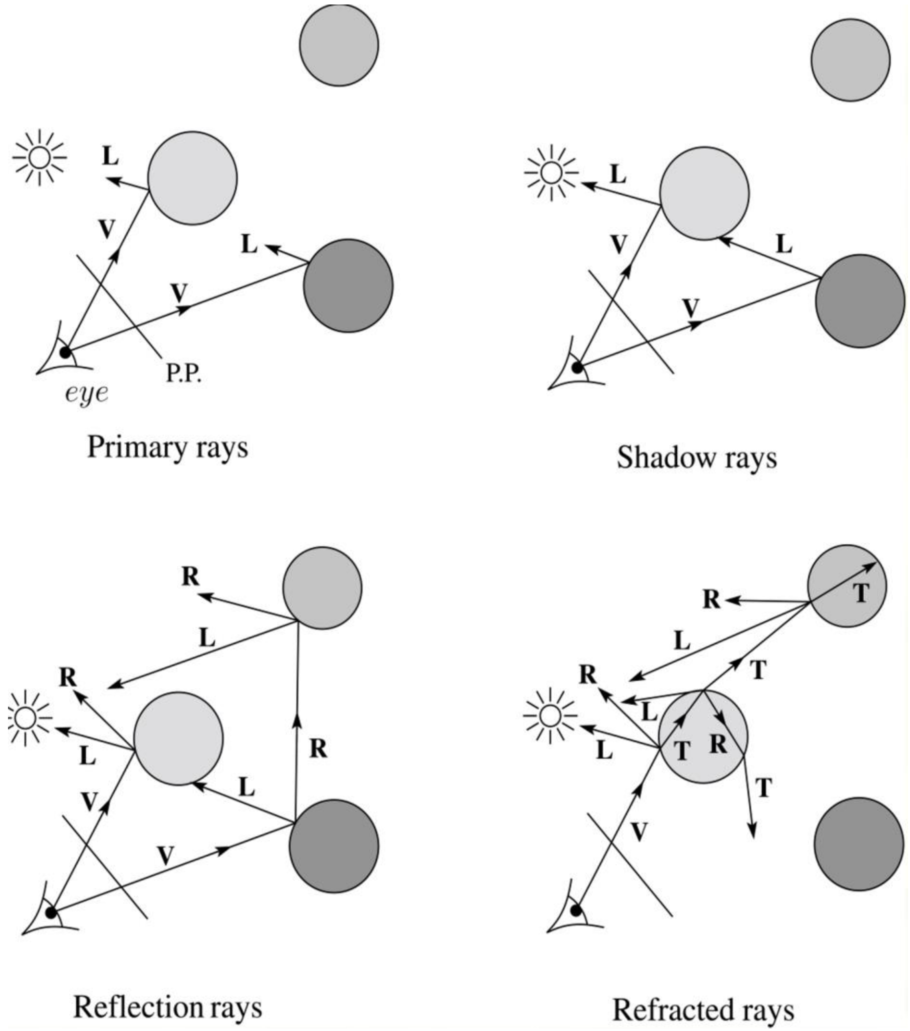
Whitted ray-tracing algorithm

- In 1980, Turner Whitted introduced ray tracing to the graphics community
  - Combines eye ray tracing + rays to light
  - Recursively traces rays



- Algorithm
  - For each pixel, trace a primary ray in direction V to the first visible surface
  - For each intersection, trace secondary rays:
    - Shadow rays in directions L to light sources
    - Reflected ray in direction R
    - Refracted ray or transmitted ray in direction T

Whitted algorithm



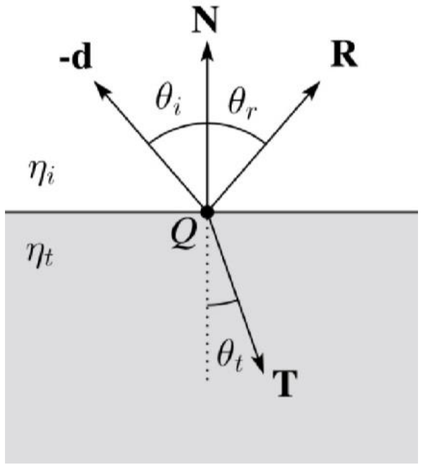
Primary rays

Shadow rays

Reflection rays

Refracted rays

Reflection and transmission



- Law of reflection:

$$\theta_i = \theta_r$$

- Snell's law of refraction

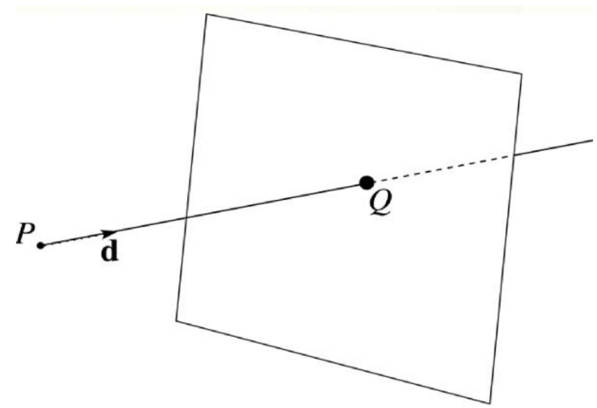
$$\eta_{\mathrm{i}} \sin\theta_{\mathrm{I}} = \eta_{\mathrm{t}} \sin \theta_{\mathrm{t}}$$

- Where η_{i} , η_{t} are **indices of refraction**

Ray-plane intersection

- We can write down the plane equation

$$a \cdot x + b \cdot y + c \cdot z + d = 0$$
$$n = [a; b; c]$$
$$n^T \cdot [x; y; z] + d = 0$$



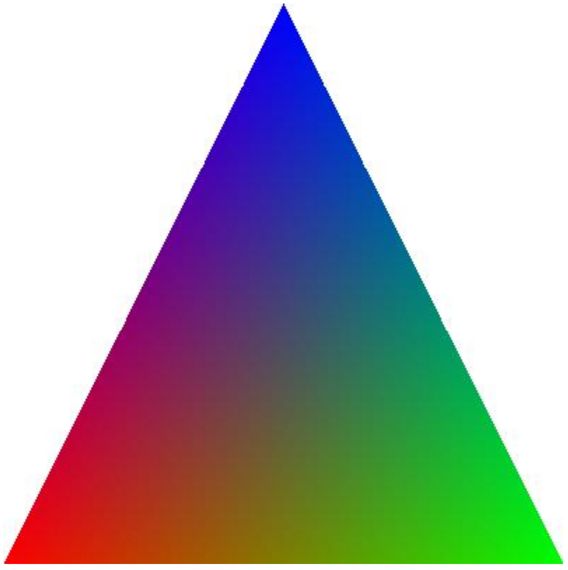
- Using parameterized line segment

$$n^T \cdot (p + td) = 0$$

- We can solve for the intersection parameter

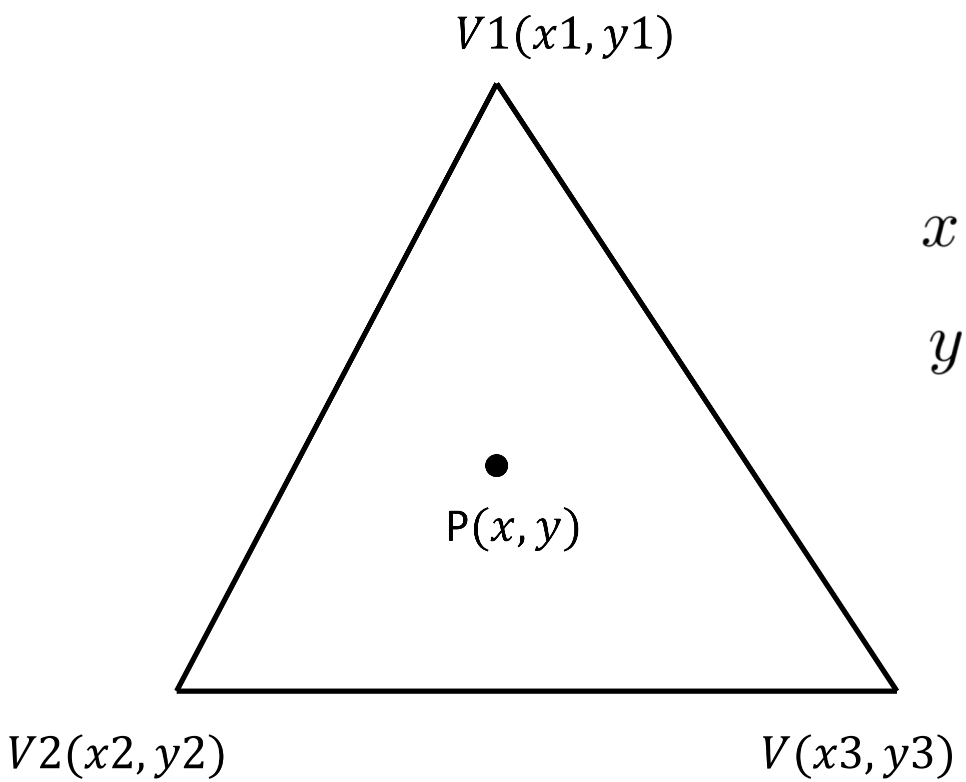
$$t = -\frac{n^T p}{n^T d}$$

Barycentric coordinates



Barycentric coordinates can be used to interpolate high-dimensional vectors

Barycentric coordinates

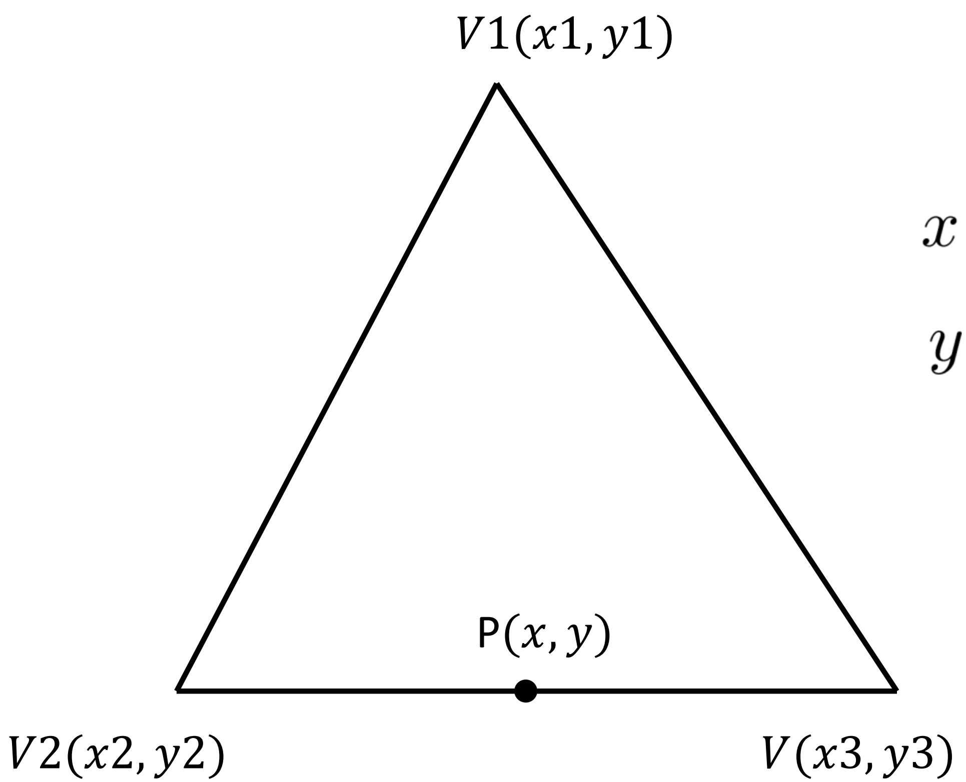


$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3$$

$$y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

Barycentric coordinates



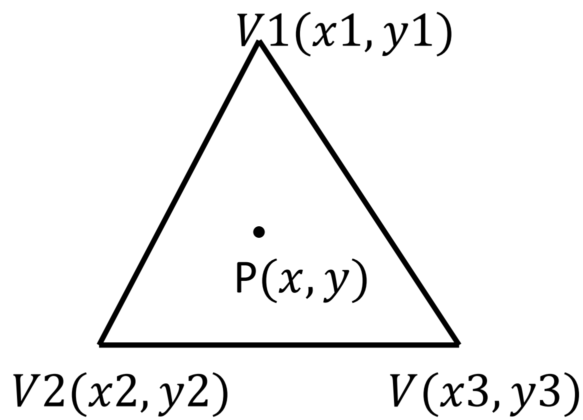
$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3$$

$$y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

$$\lambda_3 = 0$$

Computation



$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3$$

$$y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

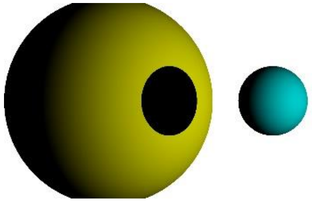
$$\lambda_1 = \frac{\det\begin{pmatrix} x & x_2 & x_3 \\ y & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}}{\det\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}} \qquad \lambda_2 = \frac{\det\begin{pmatrix} x_2 & x & x_3 \\ y_2 & y & y_3 \\ 1 & 1 & 1 \end{pmatrix}}{\det\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}} \qquad \lambda_3 = \frac{\det\begin{pmatrix} x_1 & x_2 & x \\ y_1 & y_2 & y \\ 1 & 1 & 1 \end{pmatrix}}{\det\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}}$$

Derivation is not required (yet encouraged)

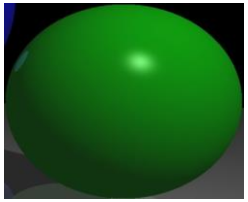
You can bring a 2-page cheat sheet to the exam

Elements of Ray Tracing

- **Shadows**

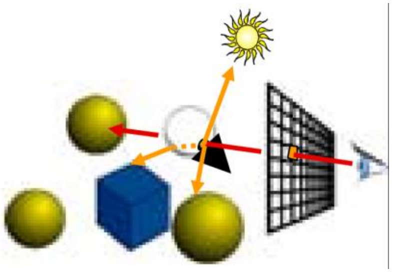


- Reflection

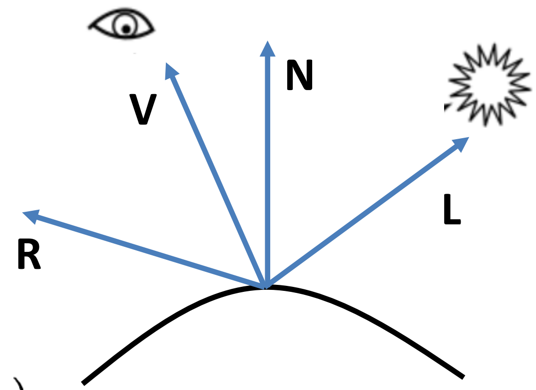


- Refraction

- Recursive Ray Tracing

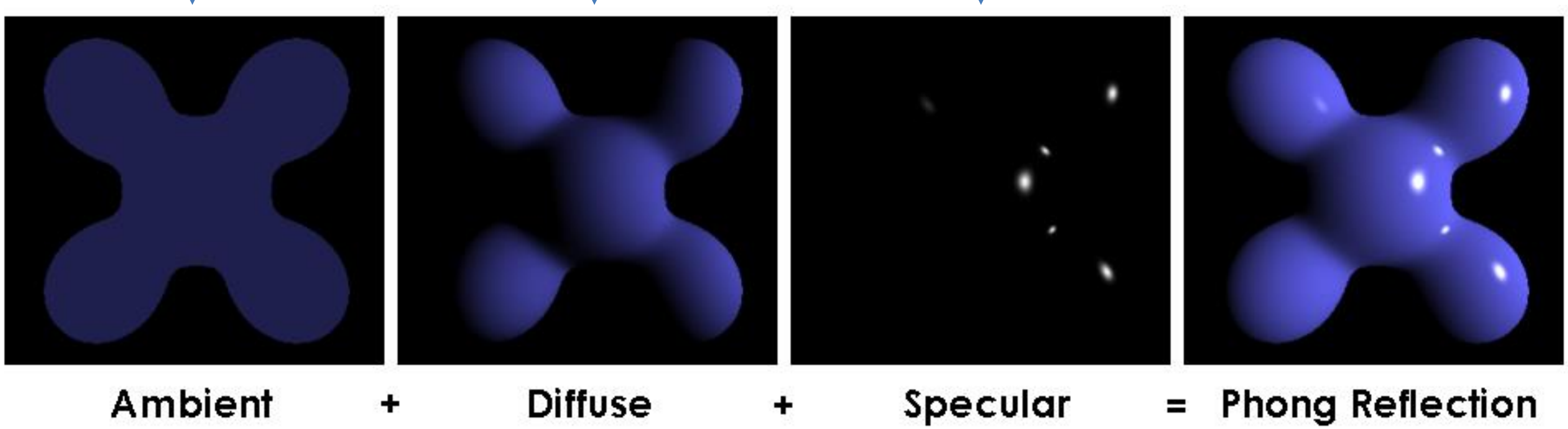


Shading --- Phong Model



$$\hat{R}_m = 2(\hat{L}_m \cdot \hat{N})\hat{N} - \hat{L}_m$$

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



More details next lecture

Image credit: https://en.wikipedia.org/wiki/Phong_reflection_model

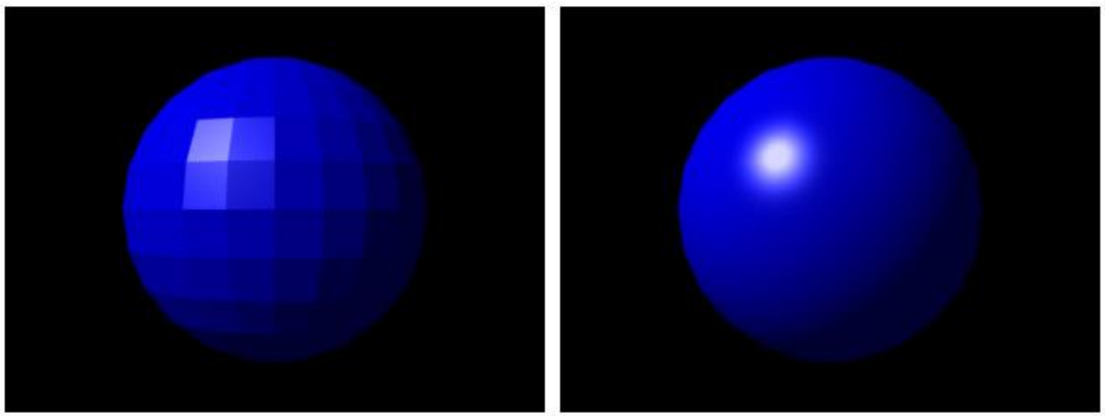
Choosing the parameters

- Experiment with different parameter settings. To get you started, here are a few suggestions
  - Try n_s in the range [0, 100]
  - Try $k_a + k_d + k_s < 1$
  - Use a small k_a (~0.1)

	n_s	k_d	k_s
Metal	large	small, color of metal	carge, color of metal
Plastic	medium	medium, color of plastic	medium, white
Planet	0	varying	0

Phong interpolation

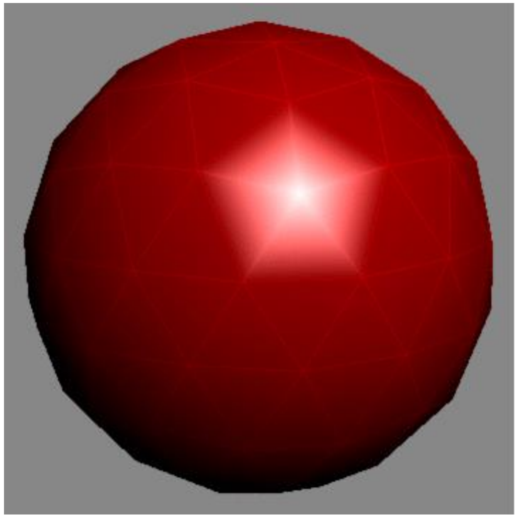
- In Phong shading a normal vector is linearly interpolated across the surface of the polygon from the polygon's vertex normal

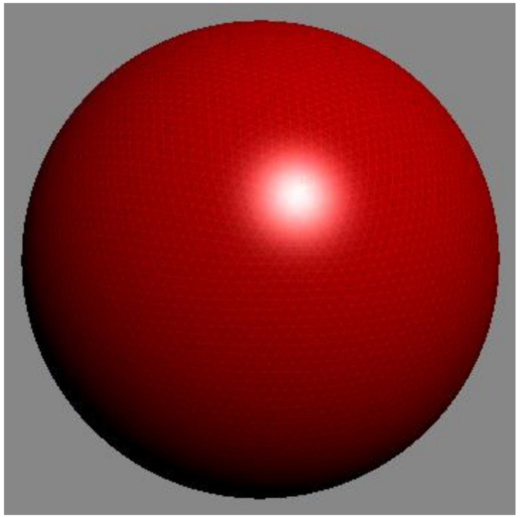


FLAT SHADING PHONG SHADING

- In contrast, Gouraud interpolation computing the lighting at the corners of each triangle and linearly interpolating the resulting colours for each pixel covered by the triangle

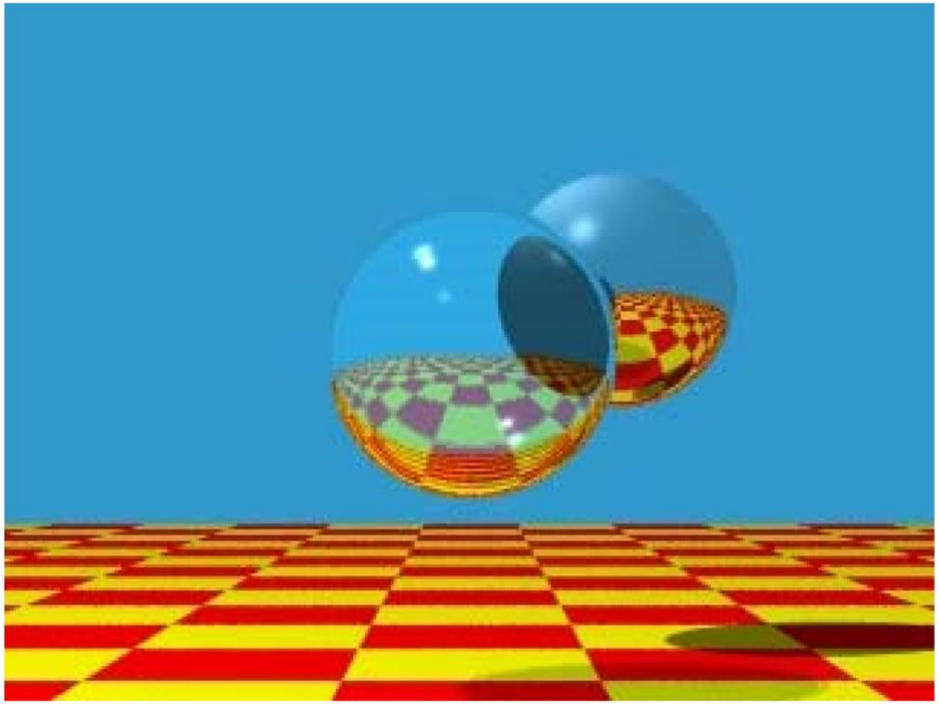
Gouraud interpolation





Gouraud interpolation is faster, while Phong interpolation is more accurate

Ray Tracing



Recursive ray tracing: Turner Whitted, 1980

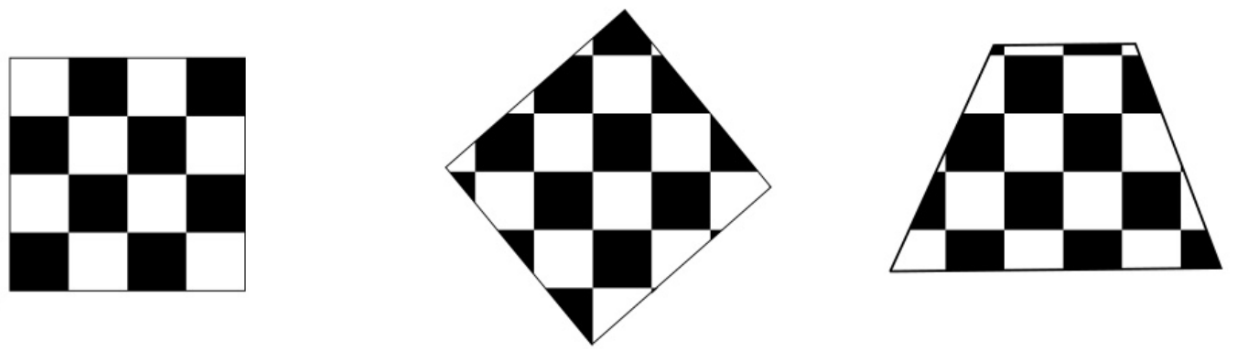
What are the rendering effects in this image?

Texture Mapping

Texture mapping (Woo et al., fig. 9-1)

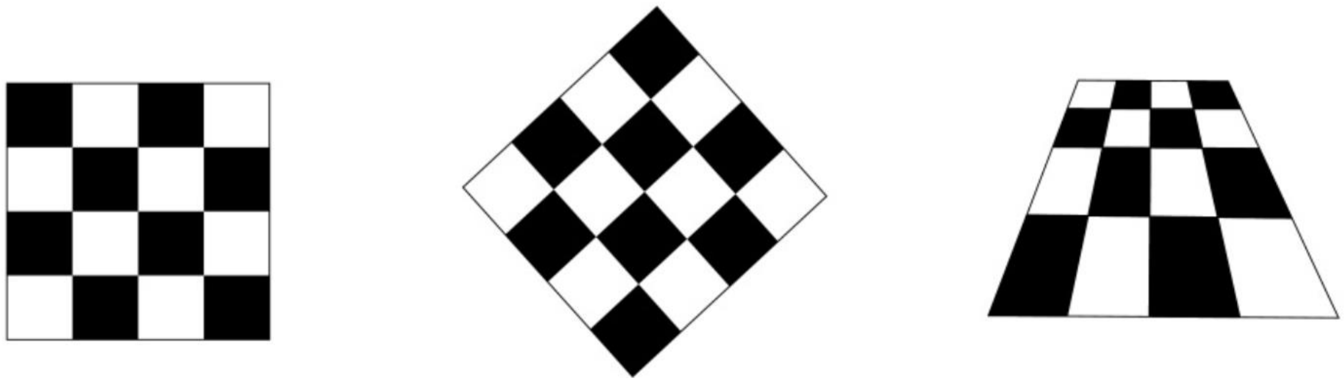
- Texture mapping allows you to take a simple polygon and give it the appearance of something much more complex
  - Due to Ed Catmull, PhD thesis, 1974
  - Refined by Blinn & Newell, 1976

Non-parametric Texture Mapping



- With "non-parametric texture mapping"
  - Texture size and orientation are fixed
  - They are unrelated to size and orientation of polygon

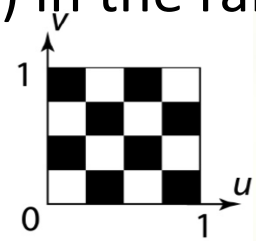
Parametric Texture Mapping



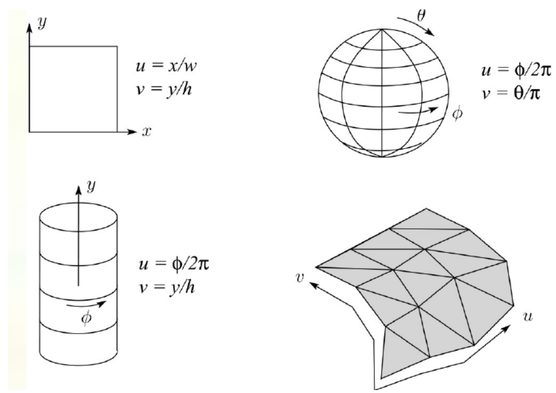
- With "parametric texture mapping," texture size and orientation are tied to the polygon
  - Separate "texture space" and "screen space"
  - Texture the polygon as before, but in texture space
  - Deform (render) the textured polygon into screen space
  - Deformation is given by parameterization
- A texture can modulate just about any parameter – diffuse color, specular color, specular exponent, …

Implementing Texture Mapping

- A texture lives in it own abstract image coordinates parameterized by (*u,v*) in the range ([0..1], [0..1]):



- It can be wrapped around many different surfaces:



- Computing (*u,v*) texture coordinates in a ray tracer is fairly straightforward
- Note: if the surface moves/deforms, the texture goes with it

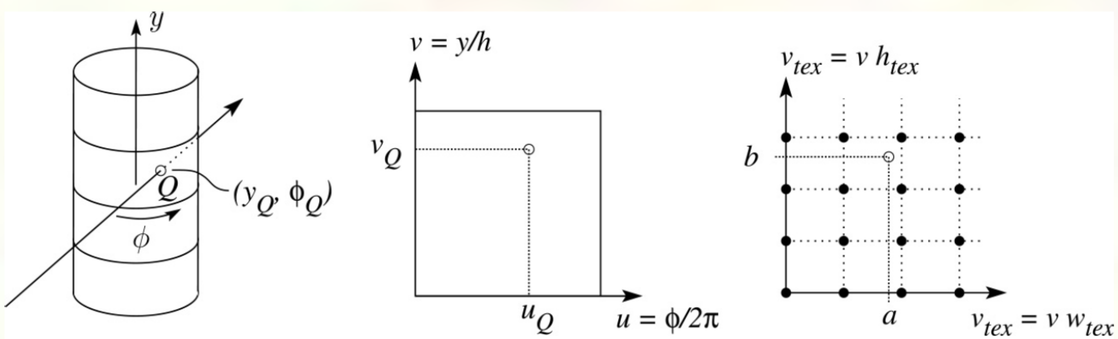
Mapping to Texture Image Coords

The texture is usually stored as an image. Thus, we need to convert from abstract texture coordinate:

(u,v) in the range $([0..1], [0..1])$

to texture image coordinates:

(u_{tex}, v_{tex}) in the range $([0..\ w_{tex}], [0..\ h_{tex}])$



Ray intersection | Mapping to abstract texture coords | Mapping to texture pixel coords

Q: What do you do when the texture sample you need lands between texture pixels?

Displacement and Bump Mapping

- Use surface offsets stored in texture
  - Perturb or dispute the surface
  - Shade on the resulting surface normals

$$\mathbf{P}(u,v)$$

$$\mathbf{S}(u,v) = \frac{\partial \mathbf{P}(u,v)}{\partial u} \qquad \mathbf{T}(u,v) = \frac{\partial \mathbf{P}(u,v)}{\partial v}$$

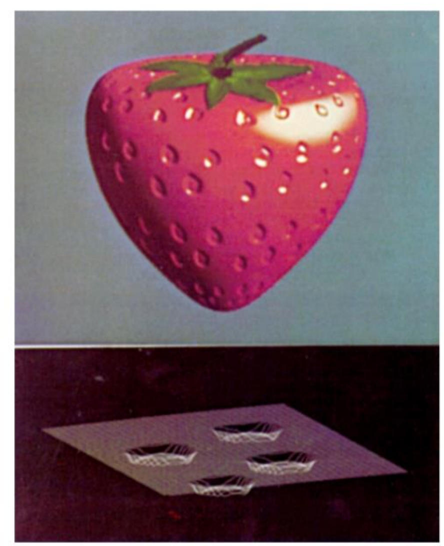
$$\mathbf{N}(u,v) = \mathbf{S} \times \mathbf{T}$$

- **Displacement**

$$\mathbf{P}'(u,v) = \mathbf{P}(u,v) + h(u,v)\mathbf{N}(u,v)$$

- **Perturbed normal**

$$\mathbf{N}'(u,v) = \mathbf{P}'_u \times \mathbf{P}'_v = \mathbf{N} + h_u(\mathbf{T} \times \mathbf{N}) + h_v(\mathbf{S} \times \mathbf{N})$$

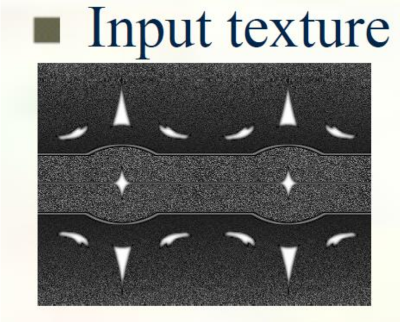


From Blinn 1976

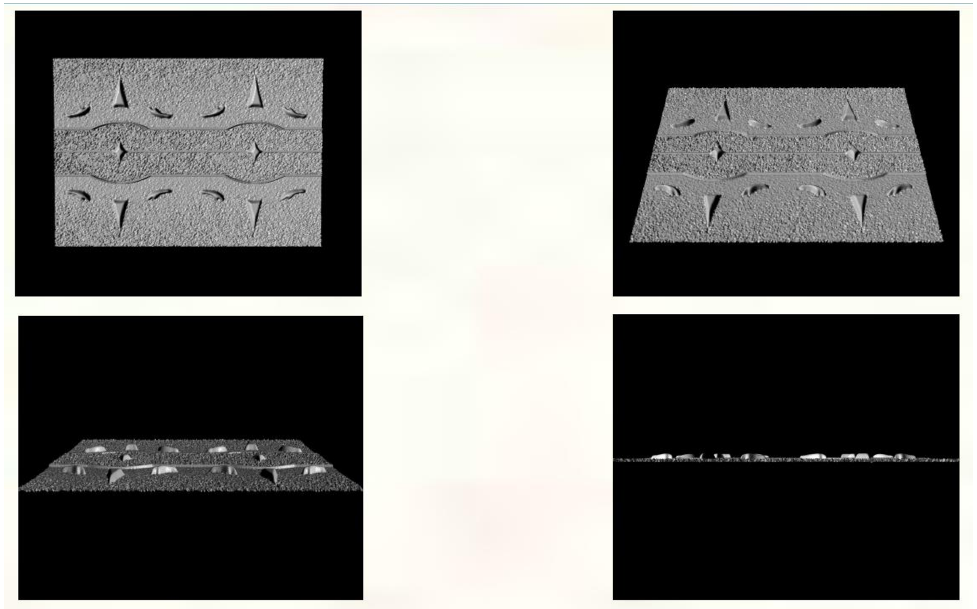
Normal Mapping

- Bump mapping via a normal map texture
  - Normal map – x,y,z components of actual normal
  - Instead of a height field 1 value per pixel
  - The normal map can be generated from the height field
  - Otherwise have to orient the normal coordinates to the surface

Displacement vs. Bump Mapping



- Rendered as displacement map over a rectangular surface



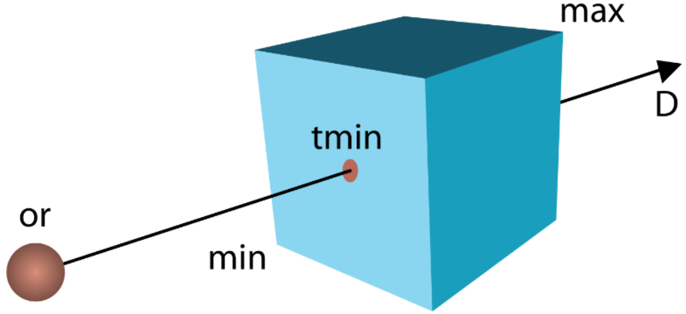
Displacement vs. Bump Mapping

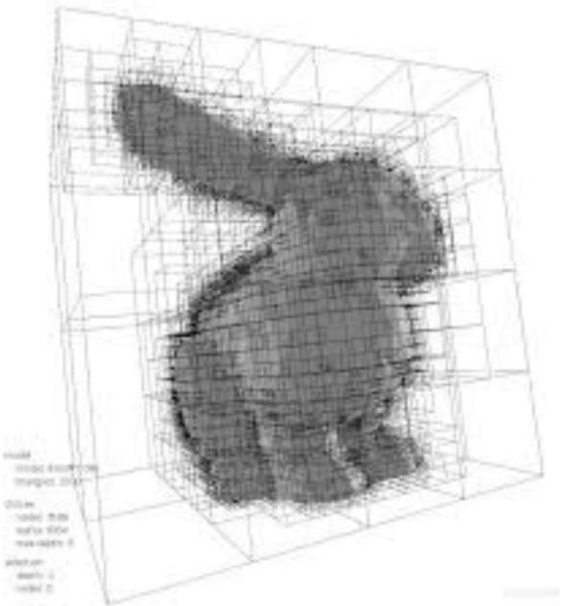
Original rendering

Rendering with
bump map wrapped
around a cylinder

Bump mapping is much faster and consumes less resources for the same level of detail compared to displacement mapping because the geometry remains unchanged.

Spatial Data Structure



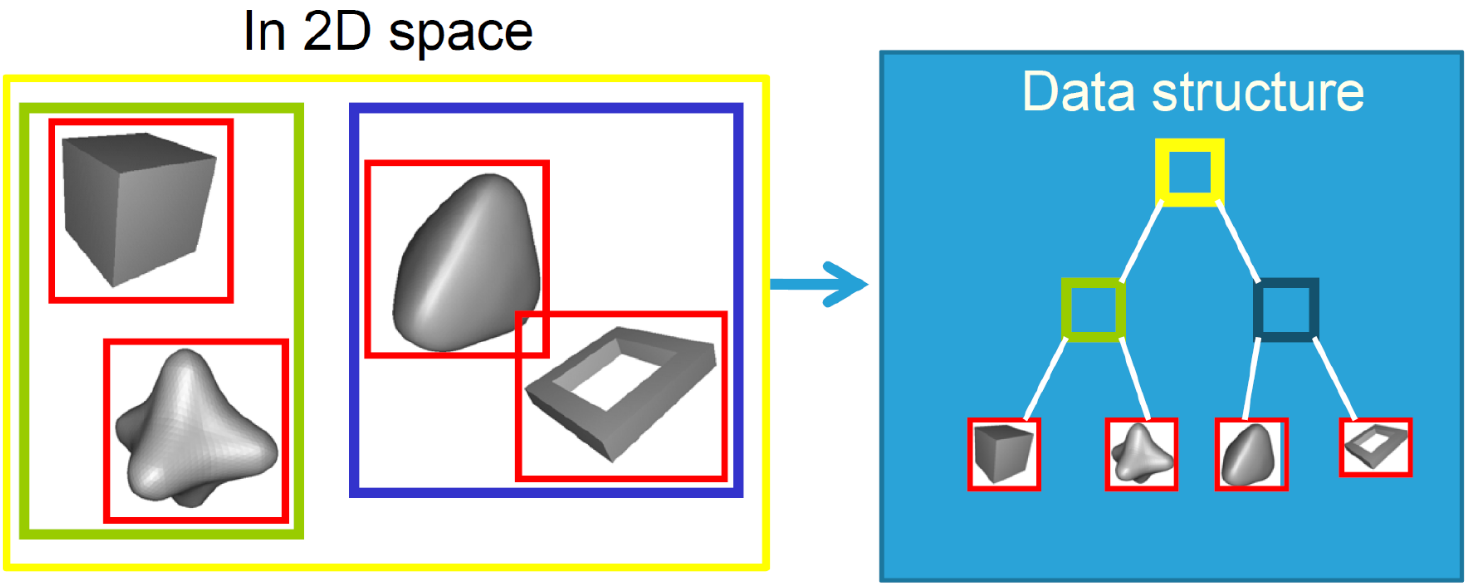


What is it?

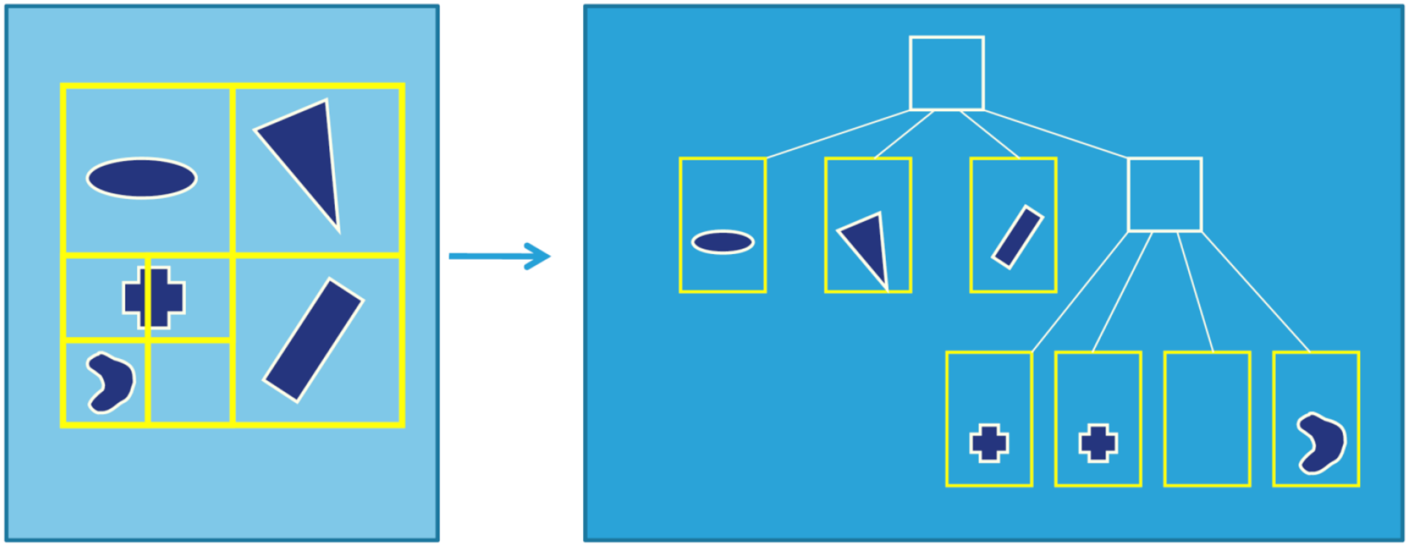
- Data structures that organize geometry (point clouds and triangular meshes) in 2D, 3D or higher dimensions
- Used for every search related problem
- Very important mathematical tool in CG
  - Ray tracing/Photon mapping
  - Collision/Intersection
  - Culling
  - Data compression
  - Level of detail
- Goal is faster processing and searching

How

- Organize geometry in a hierarchy



Octree (3D) Quadtree (2D)



- Split at half the length axis aligned
  - Always 4 children
- In 3D each square becomes a box with 8 children

Sampling

- Occurs when the sampling inherent in rendering does not contain enough information for an accurate image

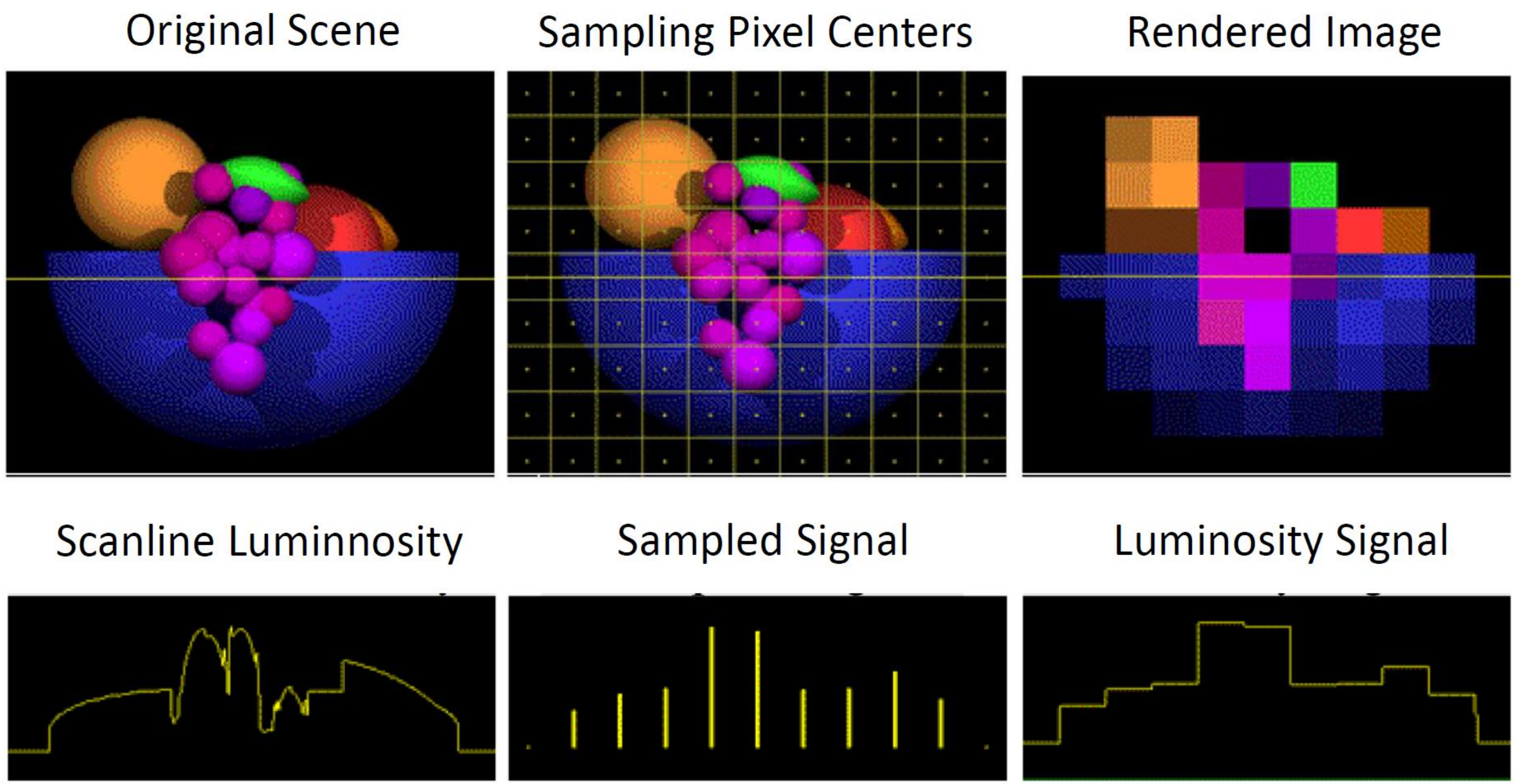


Image from SIGGRAPH 93 Educators’ Slide Set

No antialiasing

Image from SIGGRAPH 93 Educators' Slide Set

3x3 supersampling, 3x3 unweighted filter

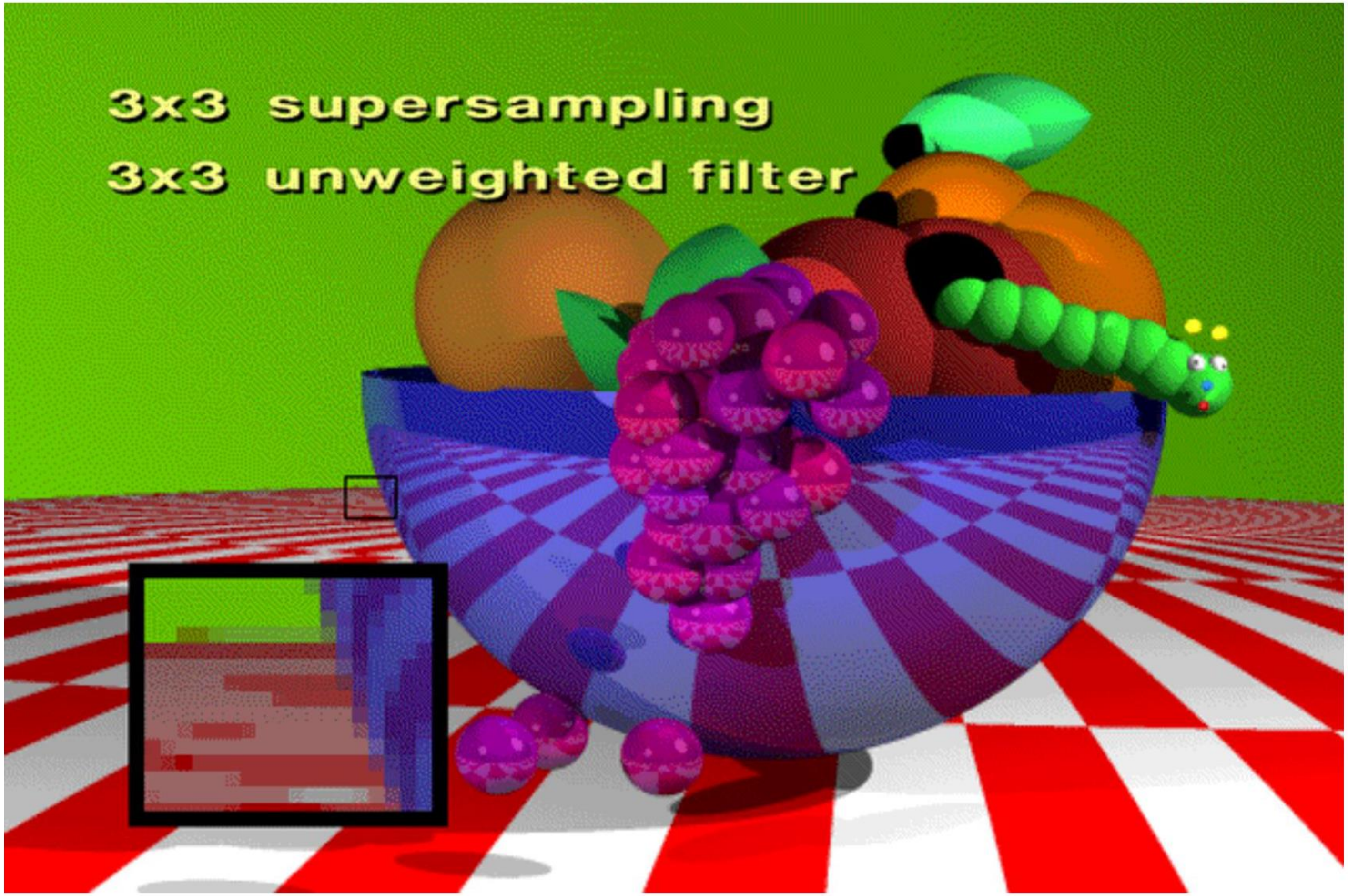
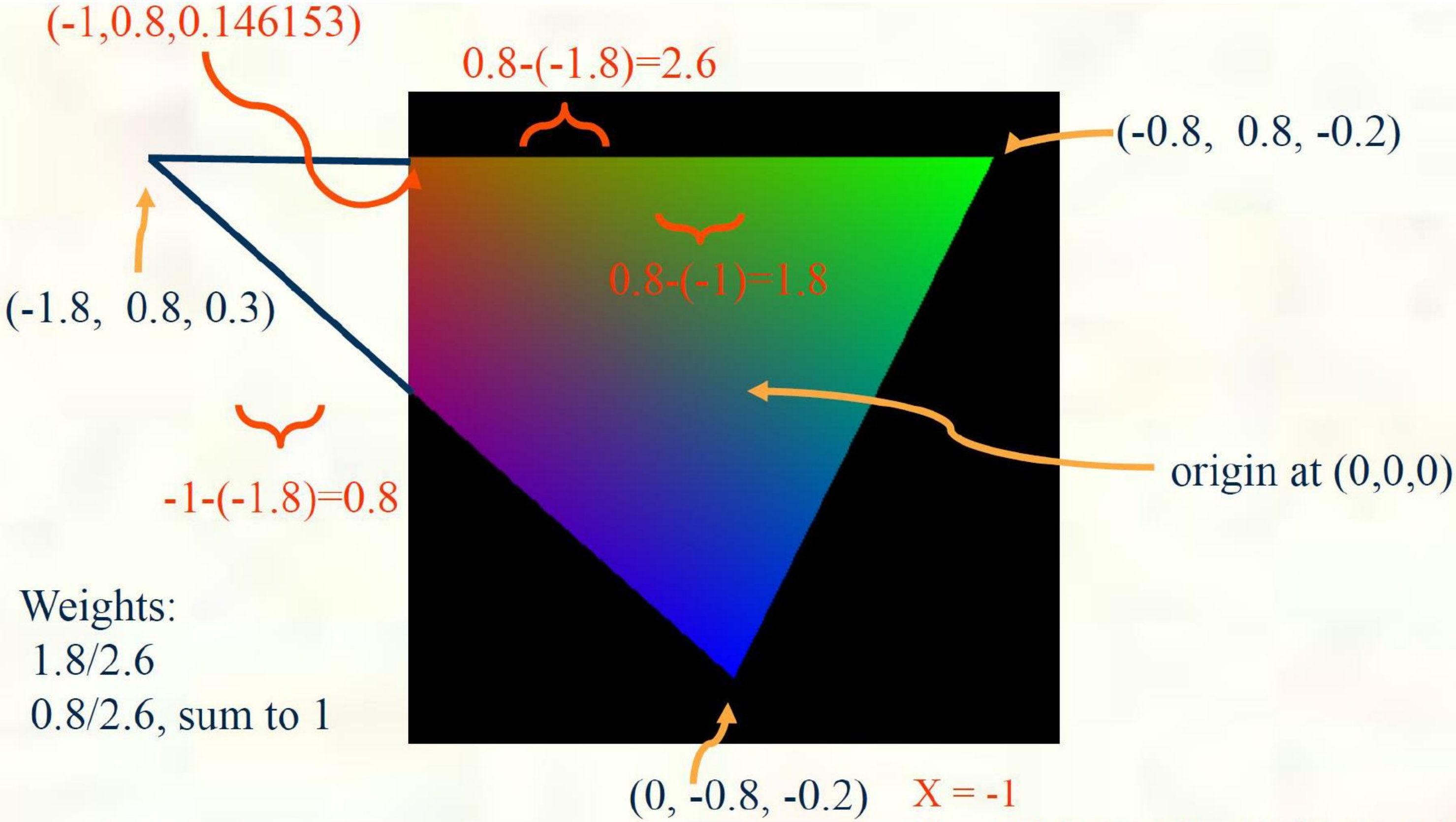


Image from SIGGRAPH 93 Educators' Slide Set

Clipping

Linear Interpolation



Weights:
1.8/2.6
0.8/2.6, sum to 1

Straightforward because all the edges are orthogonal

$X = -1$

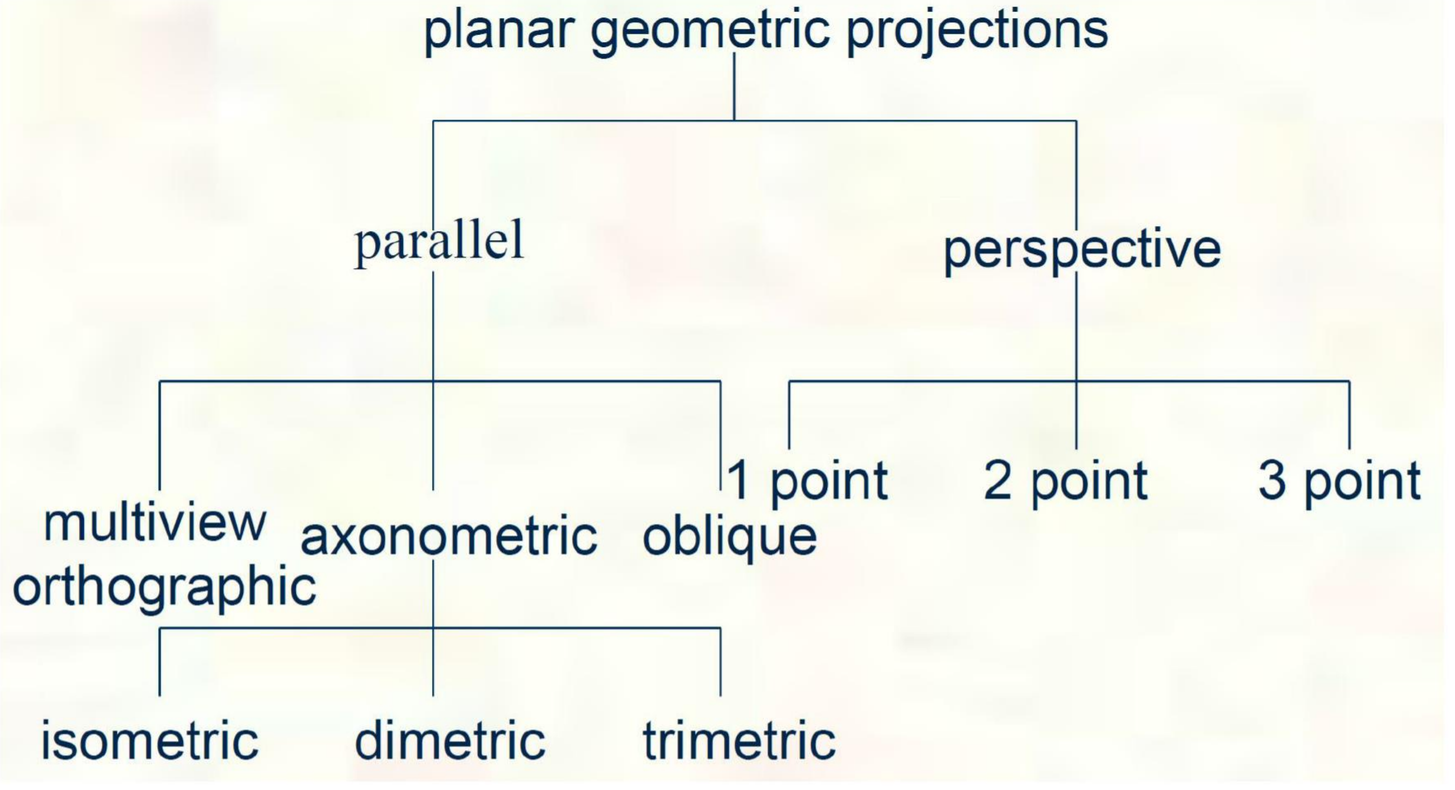
$Y = (1.8/2.6)\times 0.8 + (0.8/2.6)\times 0.8$
$= 0.8$

$Z = (1.8/2.6)\times 0.3 + (0.8/2.6)\times -0.2$
$= 0.1461538$

Clipping Complications

- Four possibilities
  - Face doesn’t actually result in any clipping of a triangle
    - Triangle is unaffected by this plane then
  - Clipping eliminates a triangle completely
    - All 3 vertices on “wrong” side of the face’s plane
  - Triangle “tip” clipped away
    - Leaving two triangles
  - Triangle “base” is clipped away
    - Leaving a single triangle
- **Strategy**: implement recursive clipping process
  - “Two triangle” case means resulting two triangles must be clipped by all remaining planes

Taxonomy of Projections

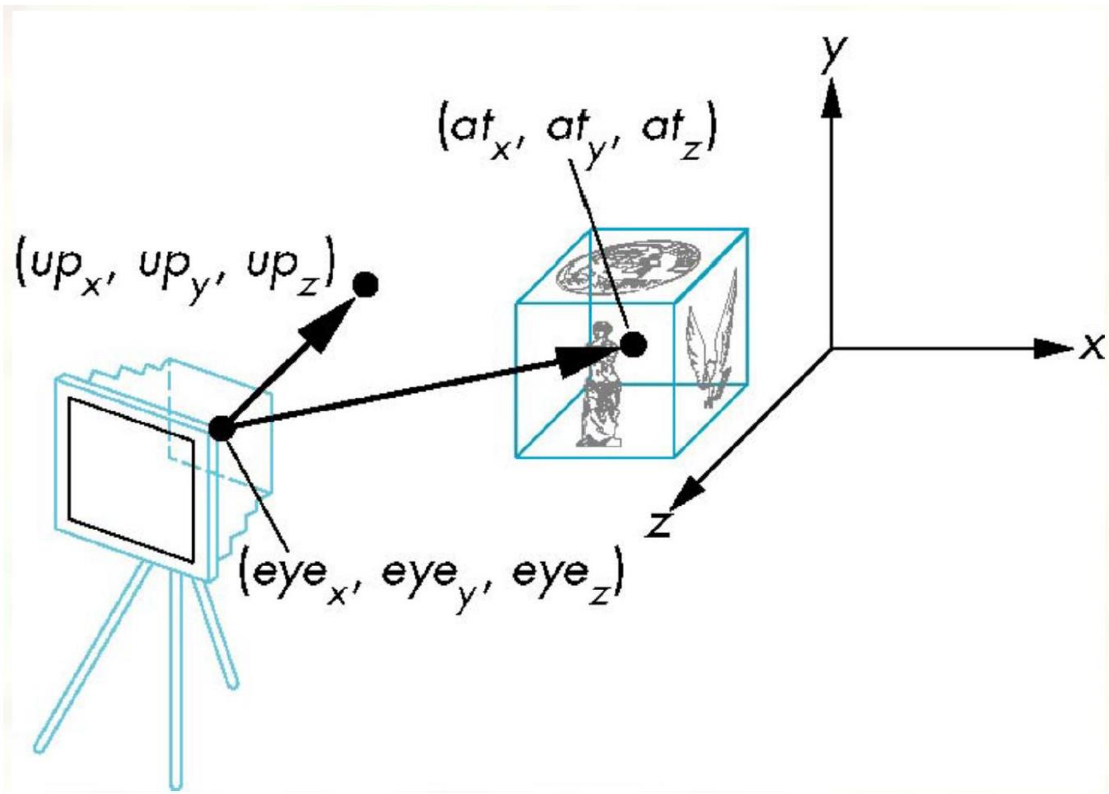


Projection

- You should be able to calculate both parallel and perspective projections
- Homogeneous coordinates

gluLookAt

gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz)



The "Look At" Algorithm

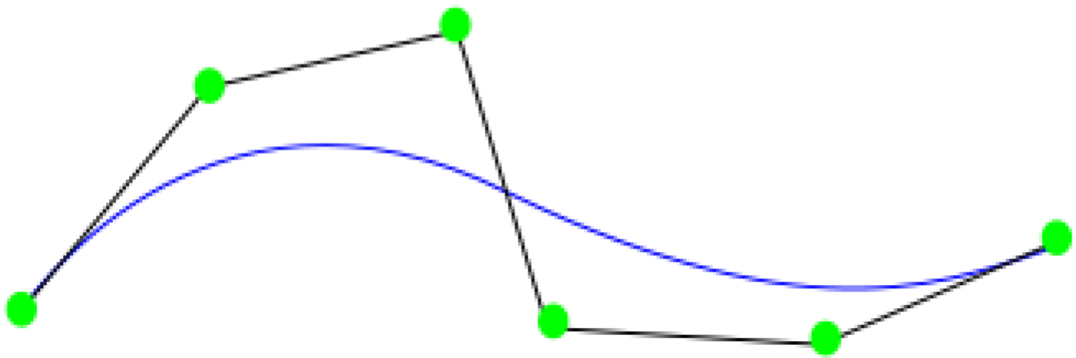
- Vector math
  - Z = eye – at
  - Z = normalize(Z) /* normalize means Z / length(Z) */
  - Y = up
  - X = Y × Z /* × means vector cross product! */
  - Y = Z × X /* orthgonalize */
  - X = normalize(X)
  - Y = normalize(Y)
- Then build the following affine 4x4 matrix

$$\begin{bmatrix} X_x & X_y & X_z & -X \bullet eye \\ Y_x & Y_y & Y_z & -Y \bullet eye \\ Z_x & Z_y & Z_z & -Z \bullet eye \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

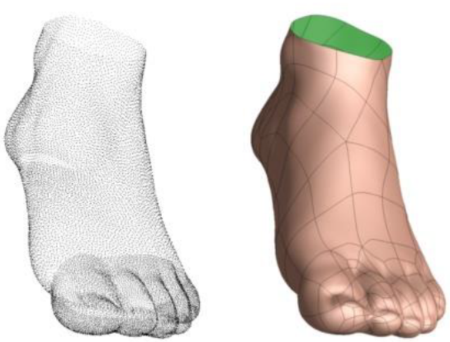
Check rounding off issues

Surface Representations

Parametric Surfaces



Bspline curve



Eck and Hoppe' 96

Implicit Surfaces

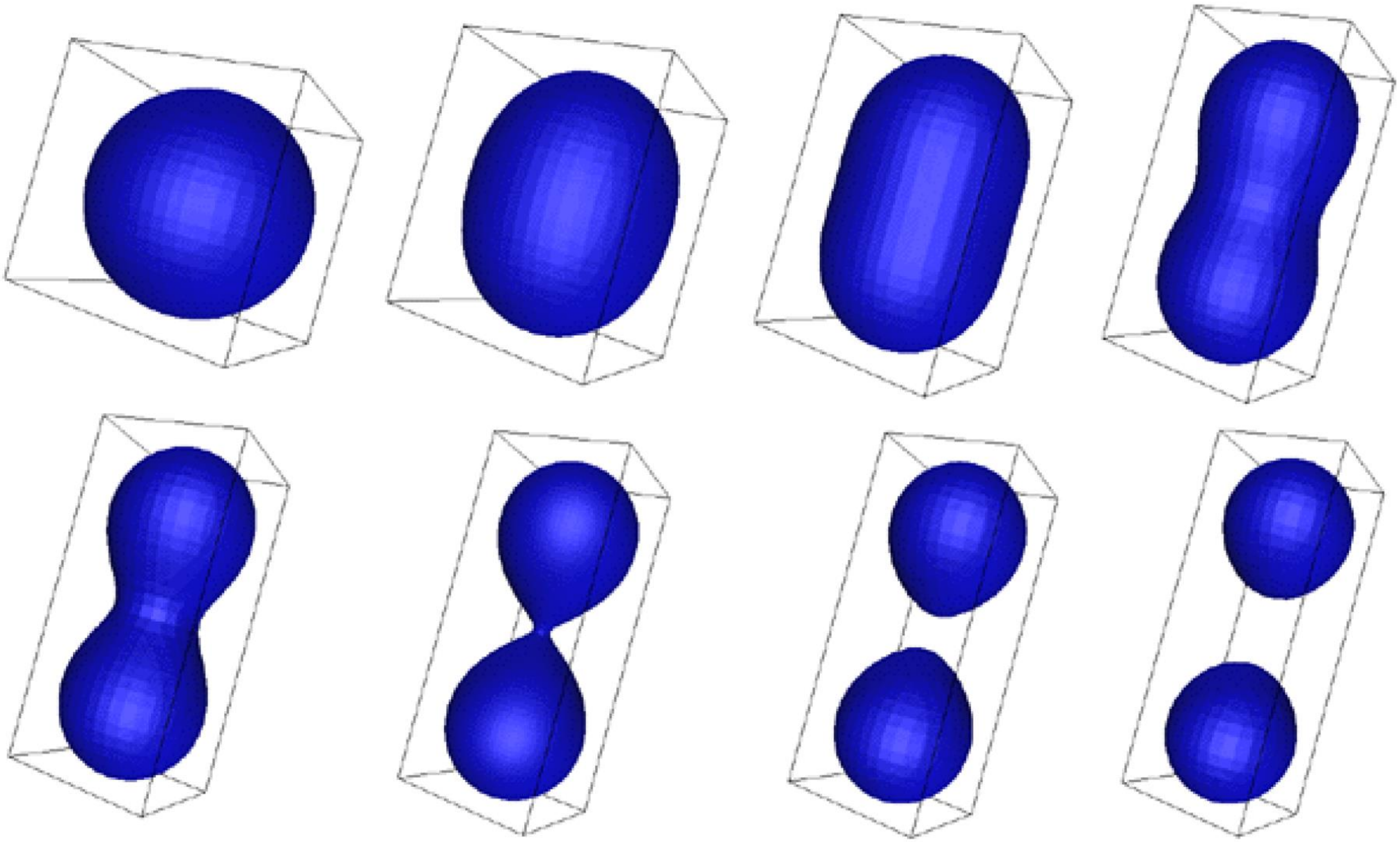
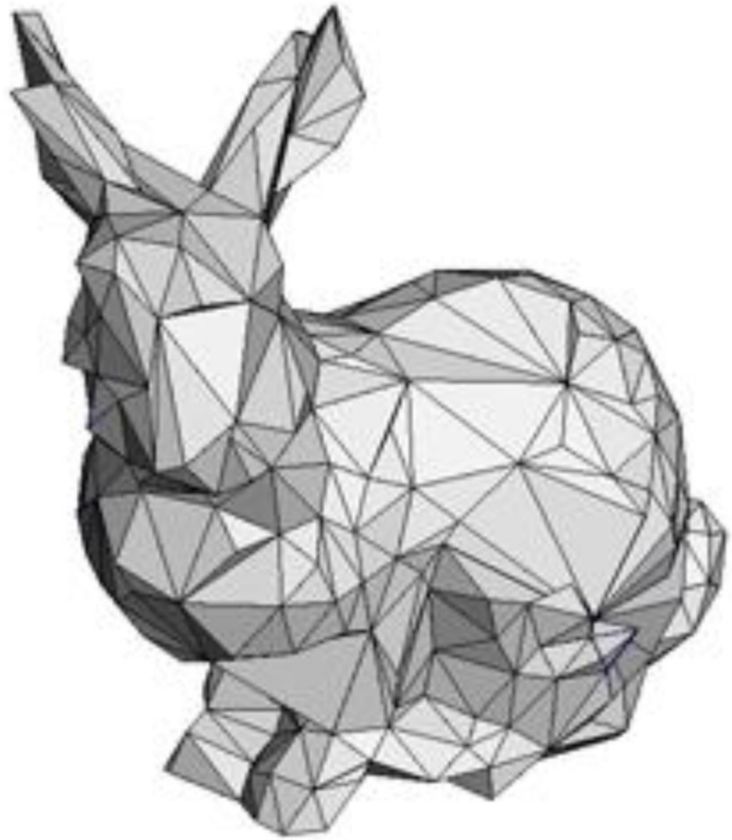


Image from http://paulbourke.net/geometry/implicitsurf/implicitsurf4.gif

Triangular Mesh



Part-based Models

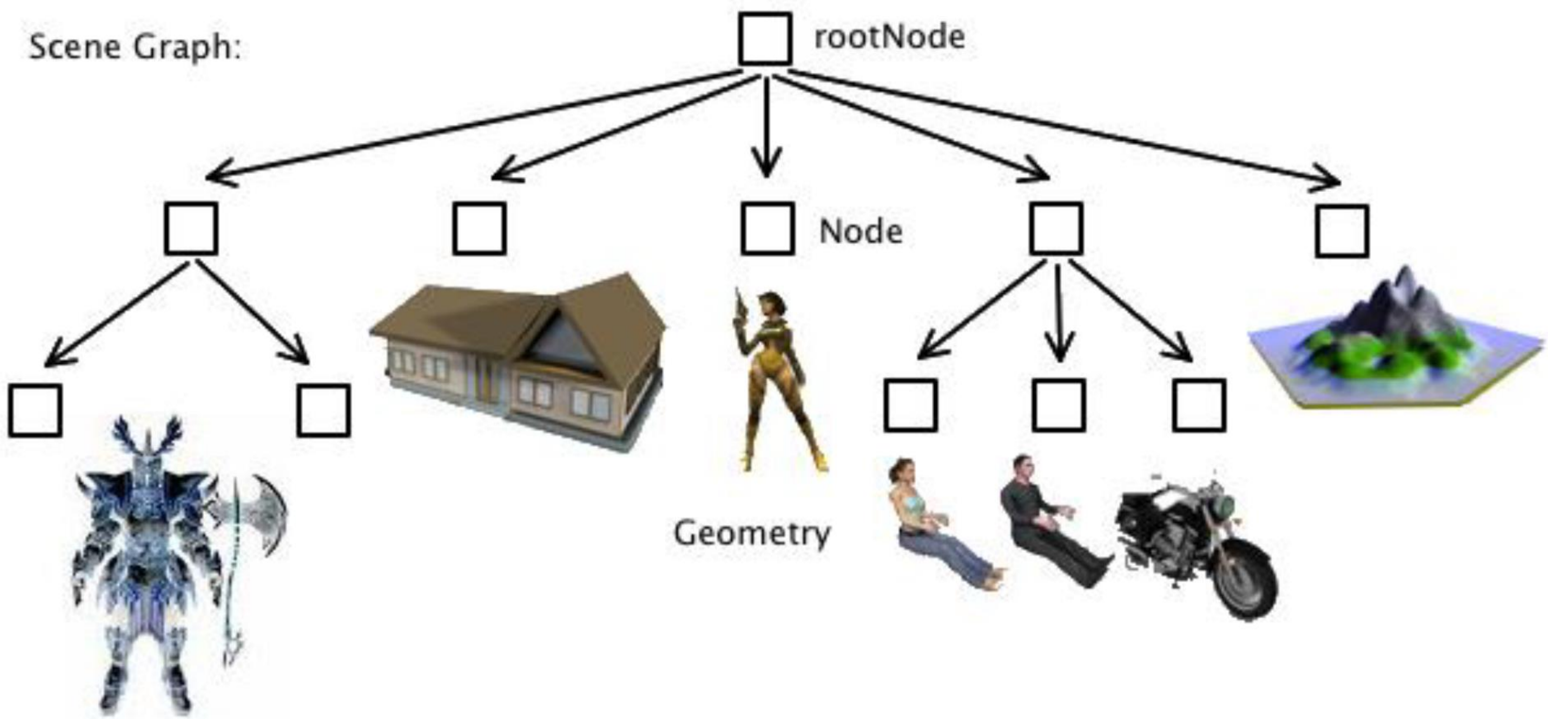


Image from https://gamedev.stackexchange.com/tags/scene-graph/info

What we need to know

- Pros and Cons of each representation
- The surface “space”
- Normal computation
- Ray-surface intersection

Hermite curves

- A cubic polynomial
- Polynomial can be specified by the position of, and gradient at, each endpoint of curve
- Determine: $x = X(t)$ in terms of x_0, x_0', x_1, x_1'

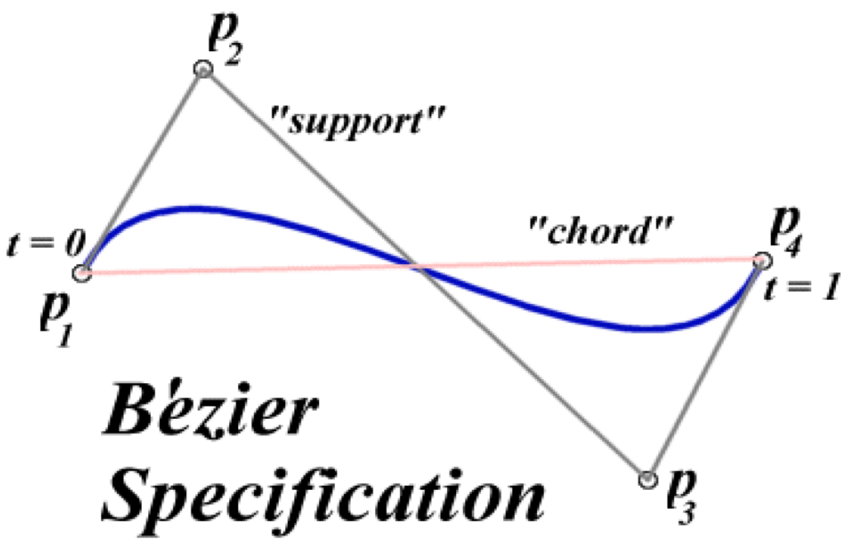
Now:

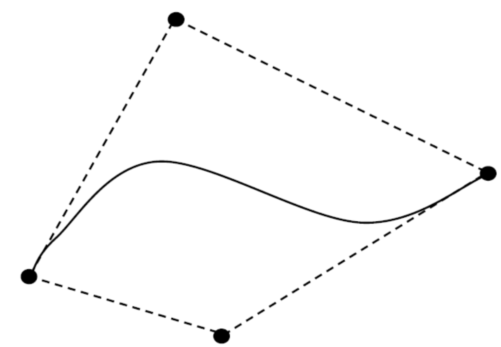
$$X(t) = a_3t^3 + a_2t^2 + a_1t + a_0$$

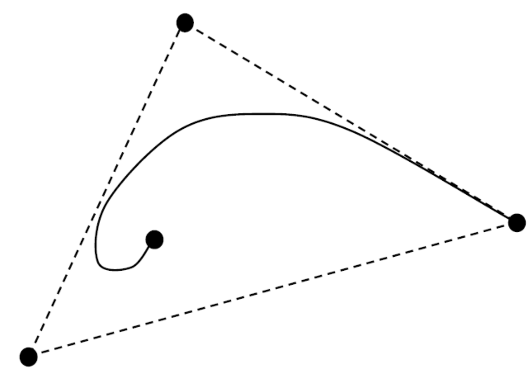
and $$X^{/}(t) = 3a_3t^2 + 2a_2t + a_1$$

Bézier Curves

- Note the Convex Hull has been shown as a dashed line – used as a bounding extent for intersection purposes

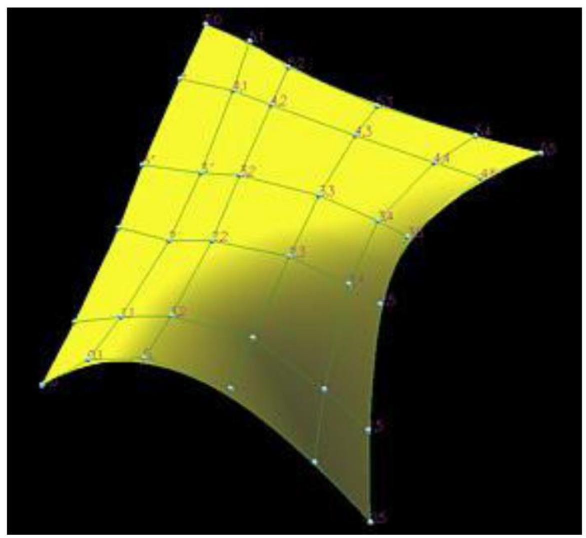






Bspline Surfaces

- The same way to we generalize Bezier curves to Bezier surfaces



$$\mathbf{p}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} N_{i,p}(u) N_{j,q}(v) \mathbf{p}_{i,j}$$

NURBS Surfaces

- General form of a NURBS curve

$$C(u) = \sum_{i=1}^{k} \frac{N_{i,n} w_i}{\sum_{j=1}^{k} N_{j,n} w_j} \mathbf{P}_i = \frac{\sum_{i=1}^{k} N_{i,n} w_i \mathbf{P}_i}{\sum_{i=1}^{k} N_{i,n} w_i}$$

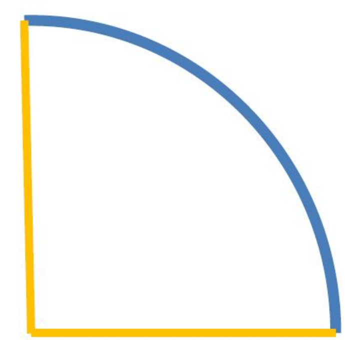
- Non-rational splines or Bezier curves may approximate a circle, but they cannot represent it exactly. Rational splines can represent any conic section, including the circle, exactly.

NURBS Representing an ARC

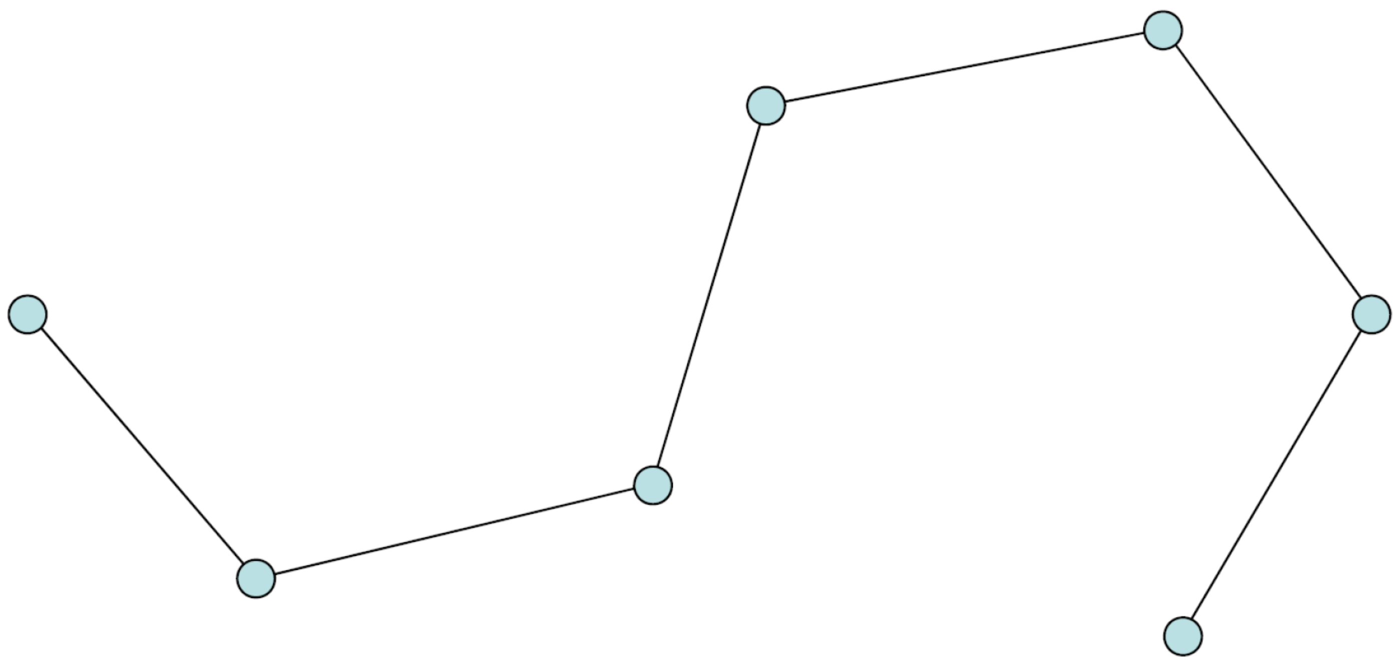
$$(x(u), y(u)) = \frac{w_0(1-u)^2(1,0) + w_1 2u(1-u)(1,1) + w_2 u^2(0,1)}{w_0(1-u)^2 + w_1 2u(1-u) + w_2 u^2}$$

$$w_0 = 1,\ w_1 = 1, \text{ and } w_2 = 2$$

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

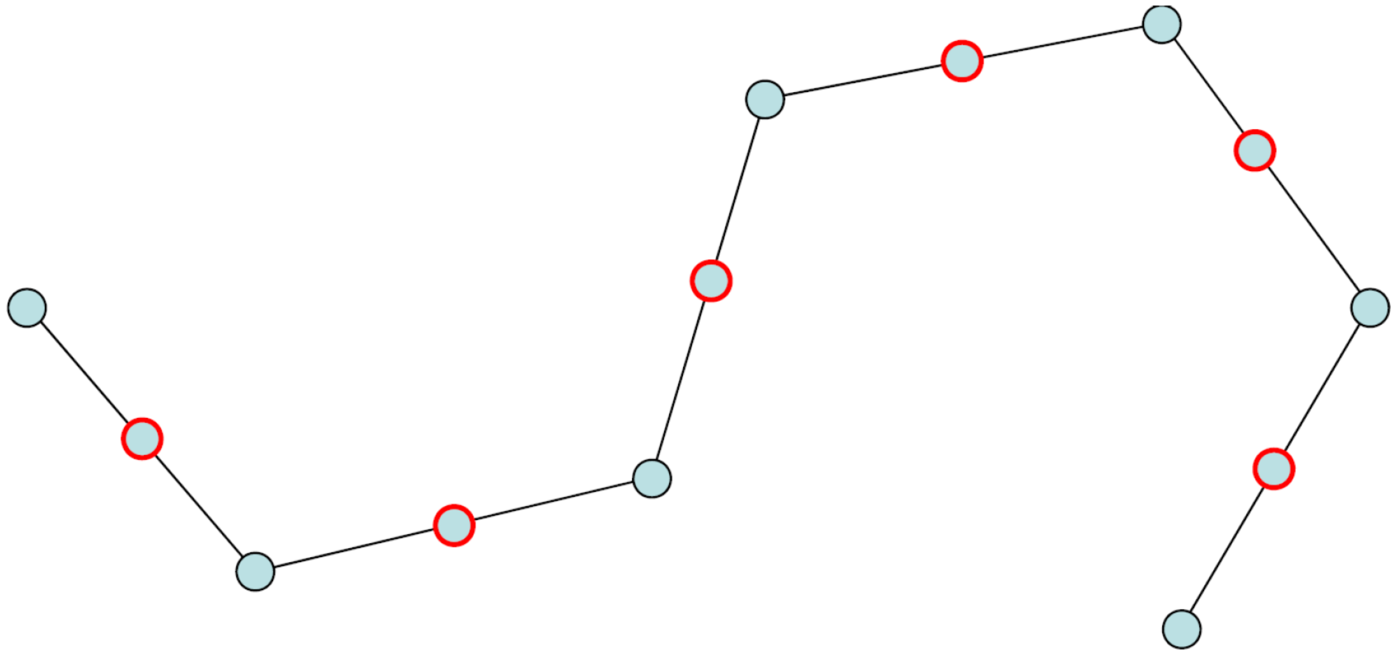


Approximating



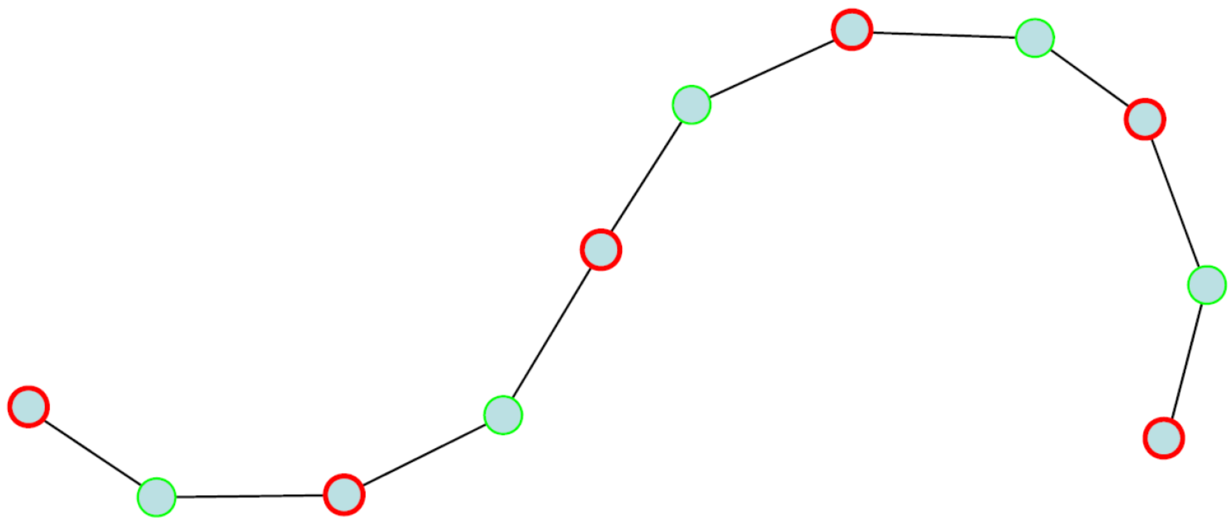
Approximating

Splitting step: split each edge in two



Approximating

Averaging step: relocate each (original) vertex according to some (simple) rule...



Definition of implicit surface

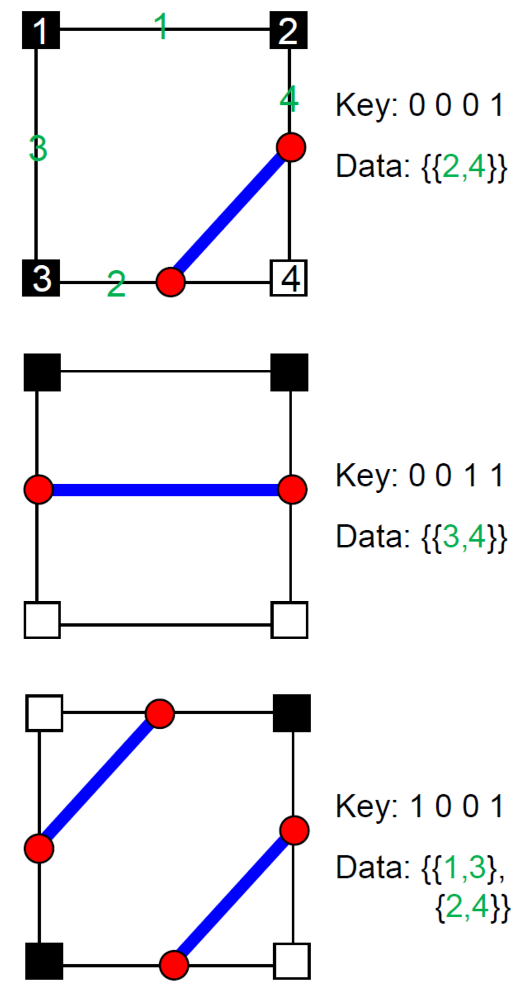
- Definition

$$\{p=(x,y,z): f(p)=0,\ p\in\mathbf{R}^3\}$$

- When *f* is algebraic function, i.e., polynomial function
  - Note that *f* and *c*f* specify the same curve
  - Algebraic distance: the value of *f(p)* is the approximation of distance from *p* to the algebraic surface *f=0*

Marching Squares (2D)

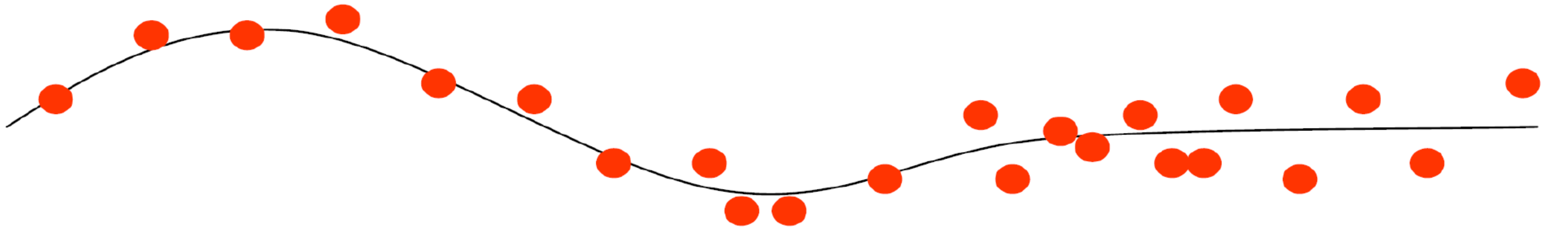
- Connecting vertices by lines
  - Lines shouldn't intersect
  - Each vertex is used once
    - So that it will be used exactly twice by the two cells incident on the edge
- Two approaches
  - Do a walk around the grid cell
    - Connect consecutive pair of vertices
  - Or, using a pre-computed look-up table
    - 2^4=16 sign configurations
    - For each sign configuration, it stores the indices of the grid edges whose vertices make up the lines.



Slide Credit: Tao Ju

Terms: Approximation/Interpolation

- Noisy data ⇒ Approximation



- Perfect data ⇒ Interpolation