CSC 317/2504: Computer Graphics

Course web site (includes course information sheet):
https://github.com/karansher/computer-graphics-csc317

Lectures: Tuesdays 18:00-20:00 on zoom
Many lecture topics videos will be pre-recorded and posted. Slides will posted on website.

Prof. Karan Singh
karan@dgp.toronto.edu
Office hours Tuesdays 17:00-18:00 on zoom or by appointment.

Tutorials: Tuesdays 20:00-21:00 on zoom

Questions:
on Assignments: Github issues pages
on Administrative stuff: Quercus

Textbooks: Fundamentals of Computer Graphics
OpenGL Programming Guide & Reference

Lecture Topics

- Introduction: What is Computer Graphics?
- Raster Images (2D image representation, manipulation and compositing) Assignment 1.
- Ray Casting (camera, visibility, normals, lighting, Phong illumination)
- Ray Tracing (shadows, supersampling, global illumination)
- Spatial Data Structures (AABB trees, OBB, bounding spheres, octree)
- Meshes (connectivity, smooth interpolation, uv-textures, subdivision, Laplacian smoothing)
- 2D/3D Transformations (Translate, Rotate, Scale, Affine, Homography, Homogeneous coordinates)
- Viewing and Projection (matrix composition, perspective, Z-buffer)
- Shader Pipeline (Graphics Processing Unit)
- Animation (kinematics, keyframing, Catmull-Romm interpolation, physical simulation)
- 3D curves and objects (Hermite, Bezier, cubic curves, curve continuity, extrusion/revolve surfaces)
- Advanced topics overview

What is Computer Graphics?

Computers:

accept, process, transform and present information.

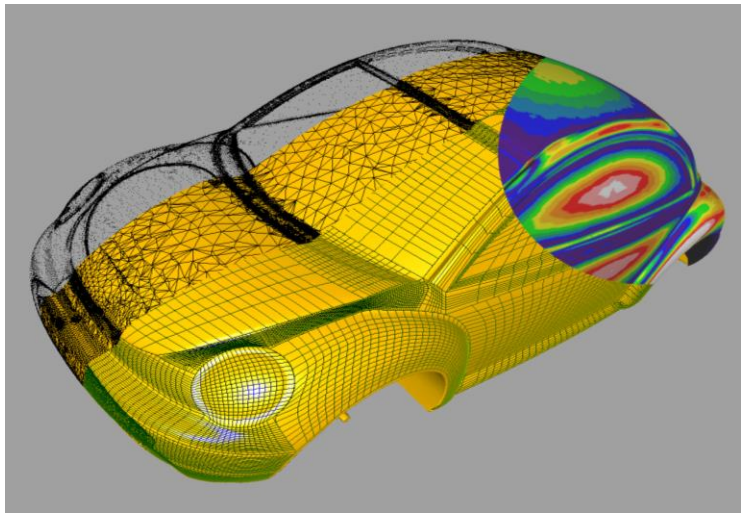
Computer Graphics:

accept, process, transform and present information

in a visual form.

“Core” Areas of Computer Graphics

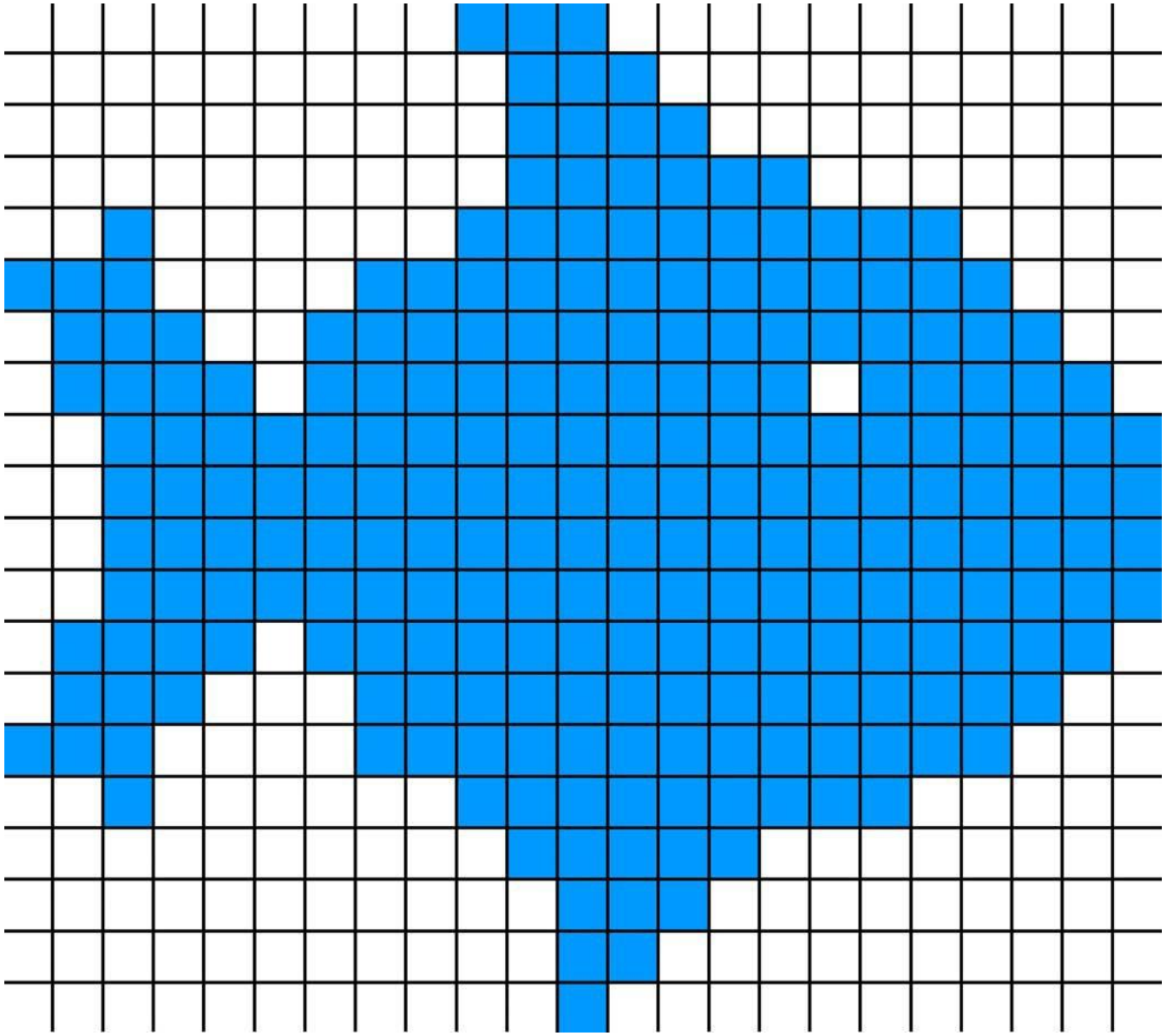
- Form (modeling)
  - How do we represent (2D or 3D) objects & scenes?
  - How do we build these representations?



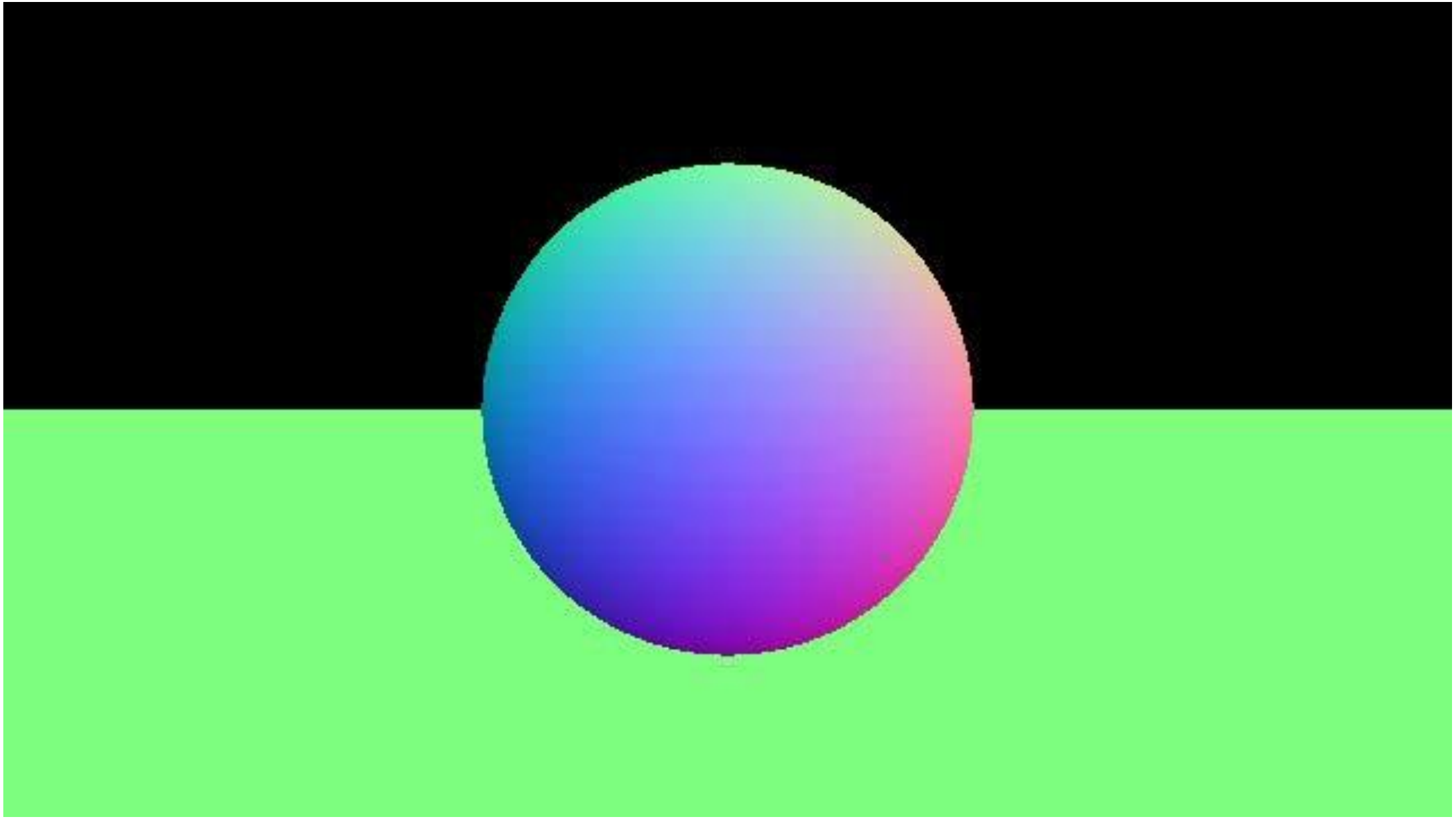
- Function, Behavior (animation)
  - How do we represent the way objects move?
  - How do we define & control their motion?

- Appearance (rendering)
  - How do we represent the appearance of objects?
  - How do we simulate the image-forming process?

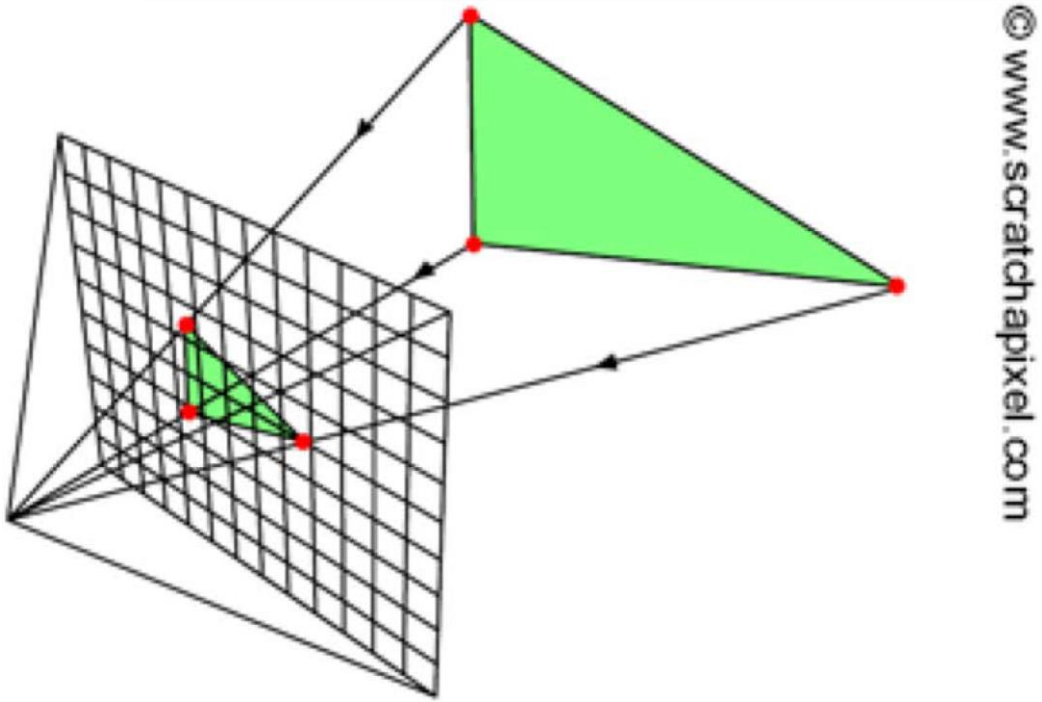
Raster Image



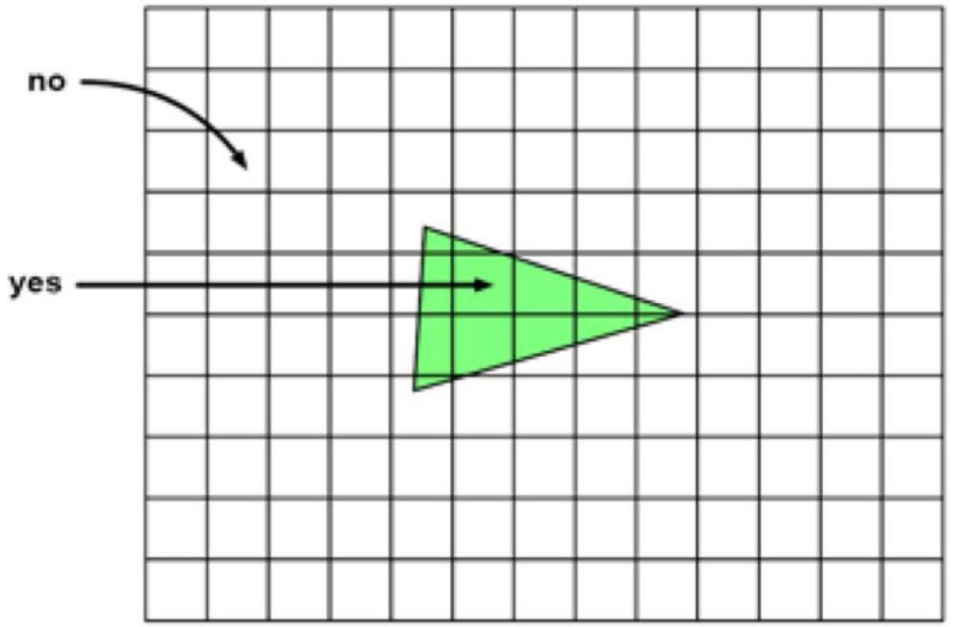
Ray Casting



Rasterization



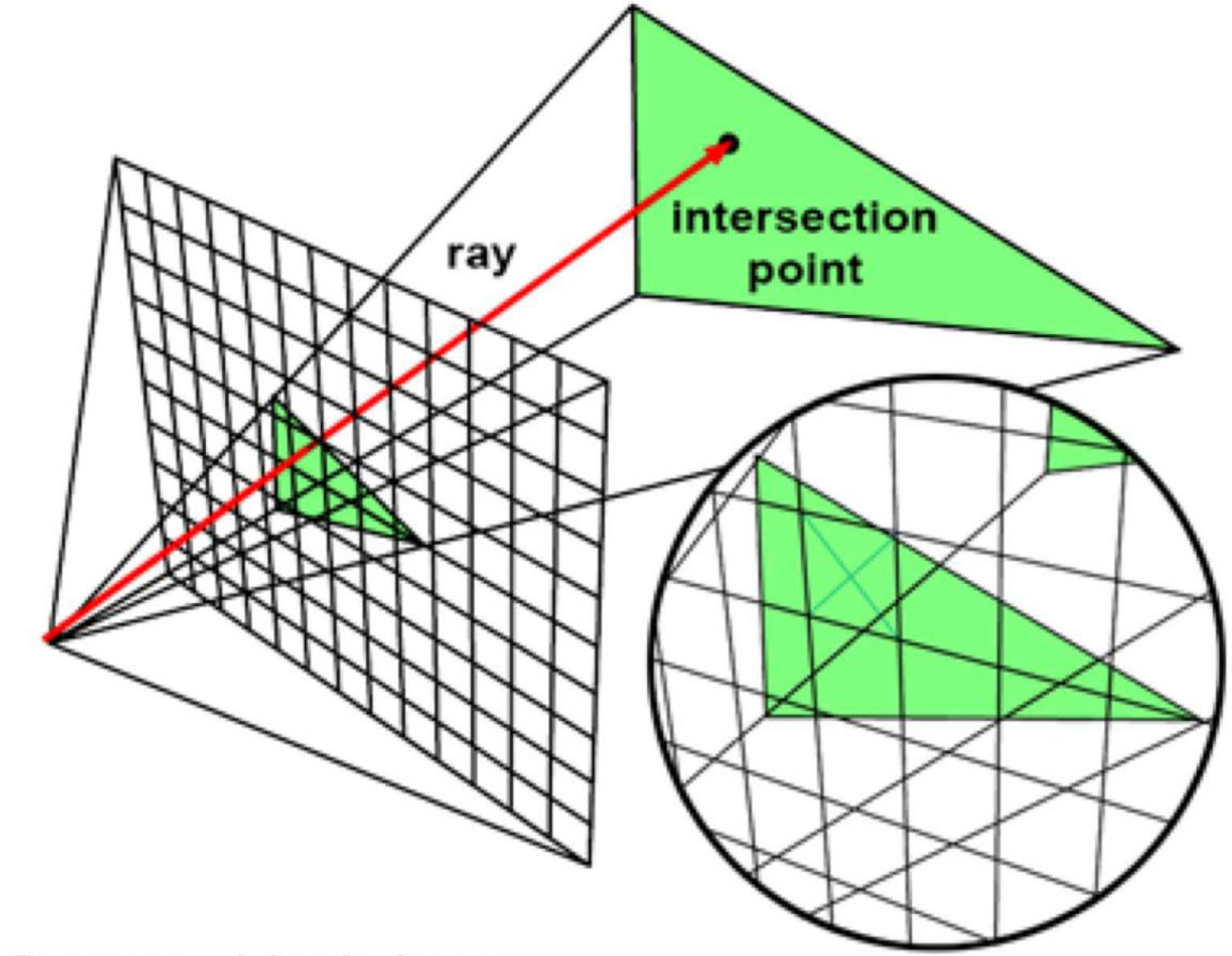
1. Project Vertices to Image Plane



2. Turn on pixels inside triangle

https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation

Ray Casting



https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation

Ray Casting vs. Rasterization

Ray Casting

```
for each image pixel {
     for each scene object {
          ...
     }
}
```

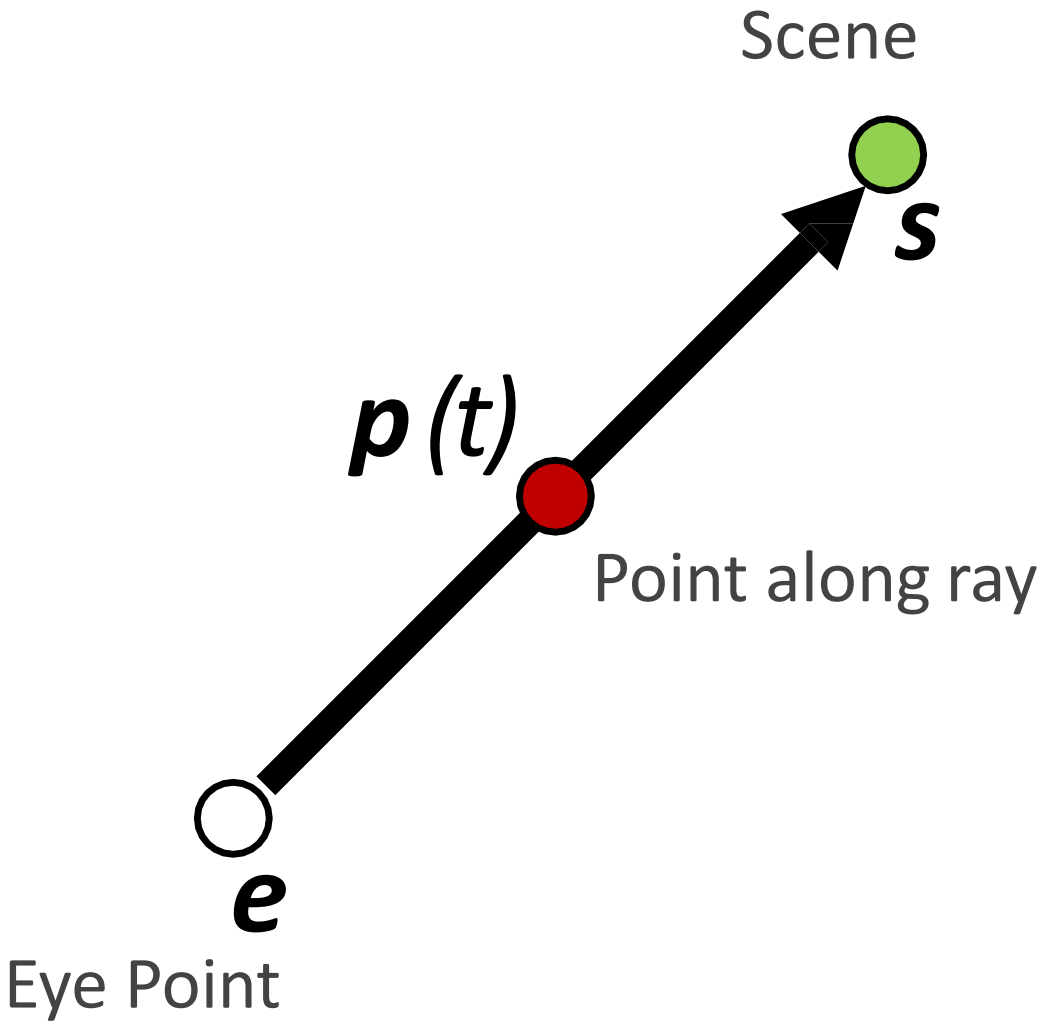
Rasterization

```
for each scene object {
     for each image pixel {
          ...
     }
}
```

Ray

$$\boldsymbol{p}(t) = \boldsymbol{e} + t(\boldsymbol{s} - \boldsymbol{e})$$

$$\boldsymbol{p}(0) = \boldsymbol{e}$$
$$\boldsymbol{p}(1) = \boldsymbol{s}$$

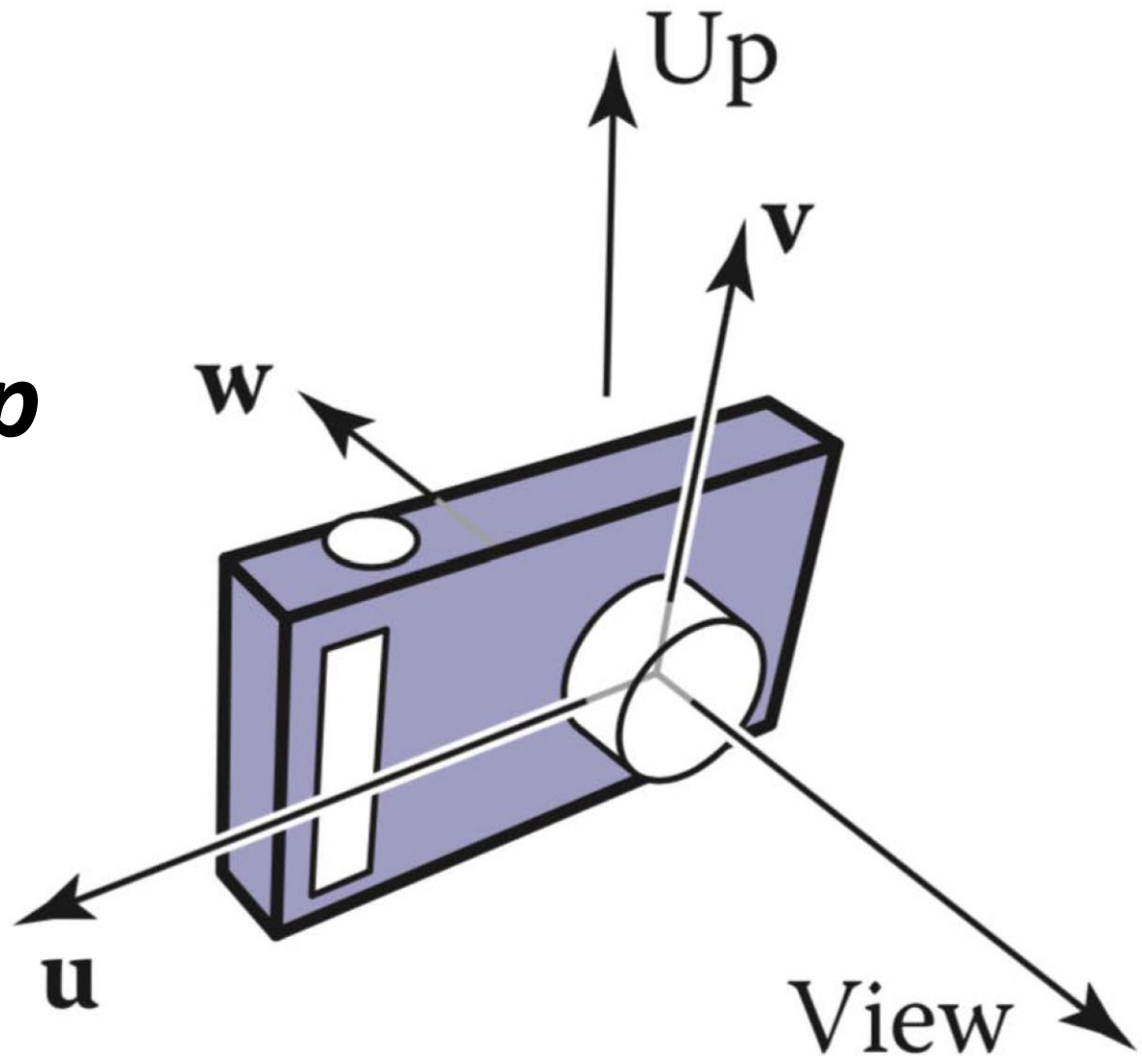


Camera

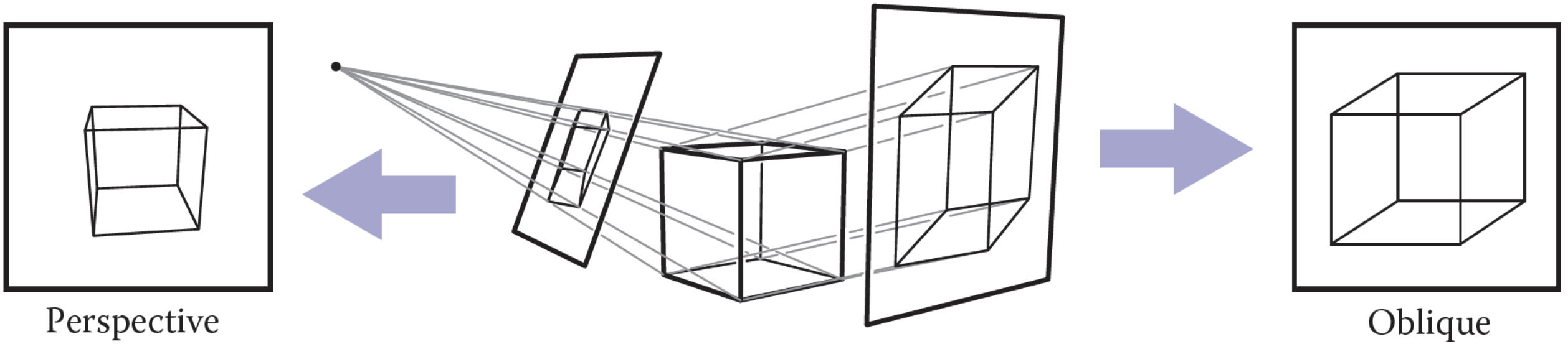
$w = -view$

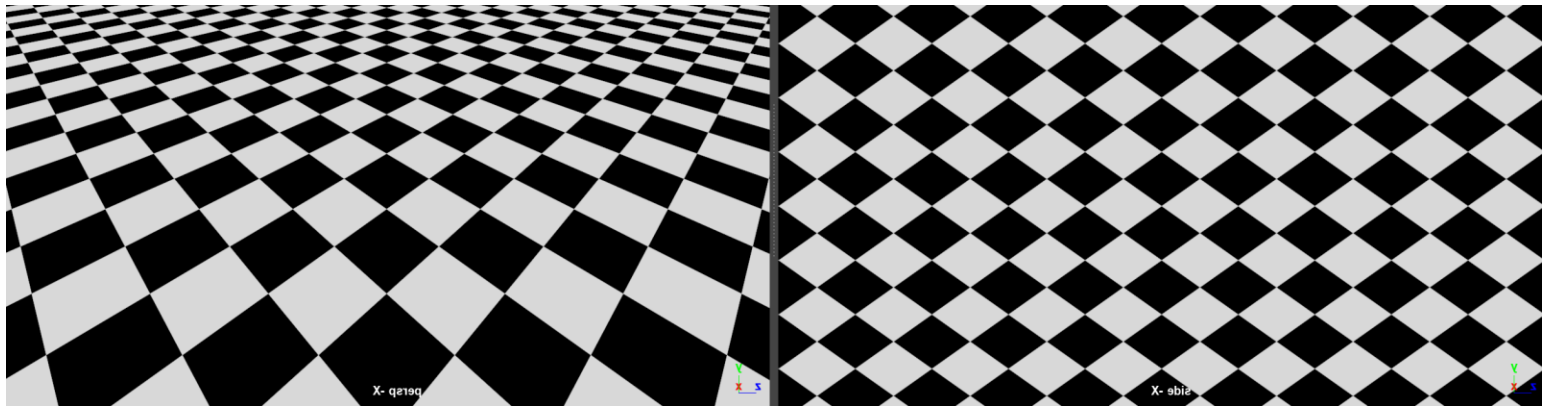
$u = view \times up$

$v = w \times u$

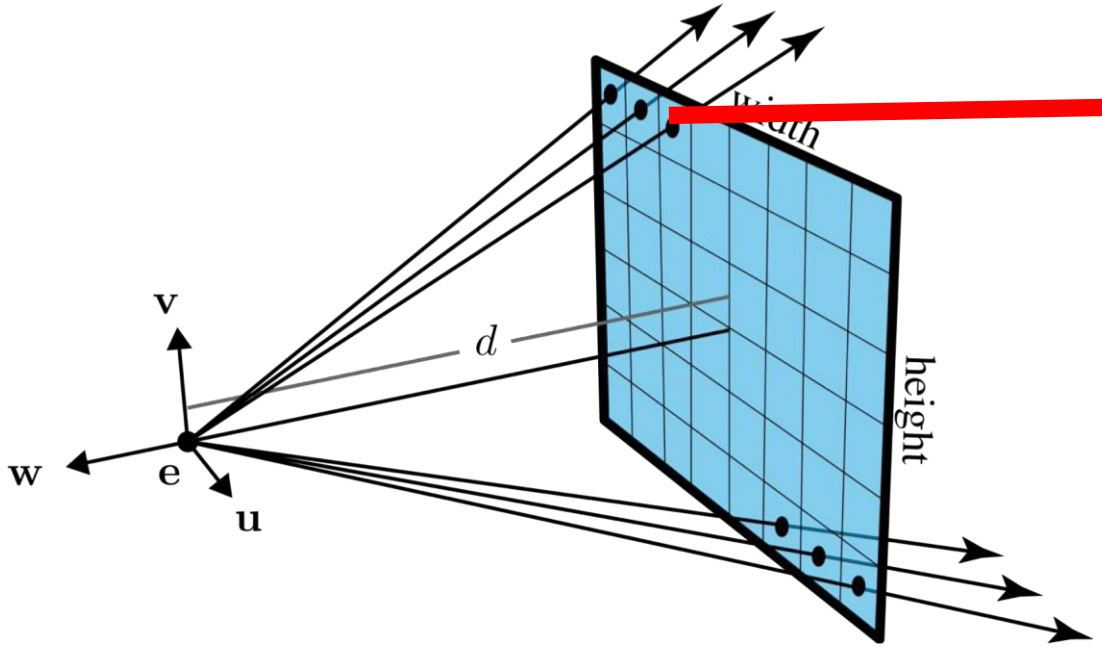


Orthographic v Perspective Projection





Ray Equation in World Space



$$\mathbf{p}(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e})$$

$$\mathbf{p}(t) = \mathbf{e} + t \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} \end{bmatrix} \begin{bmatrix} u(i) \\ v(j) \\ -d \end{bmatrix}$$

Camera Transformation Matrix

Ray-Sphere Intersection

Substitute ray equation into implicit equation for sphere

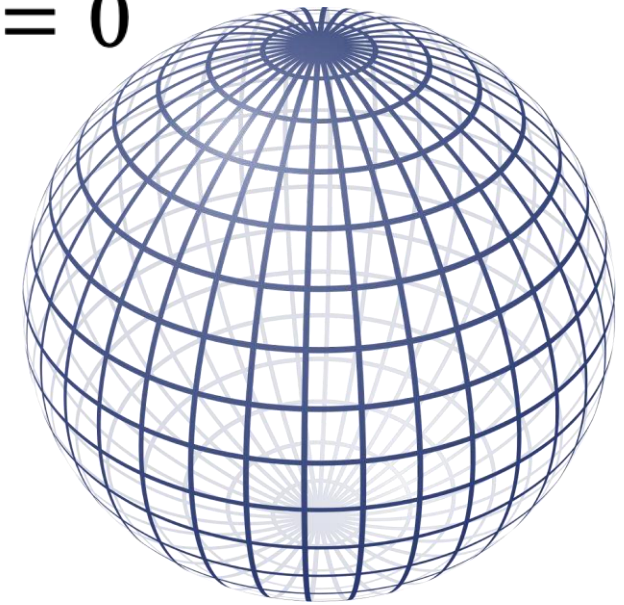
$$(\mathbf{e} + t\vec{\mathbf{d}} - \mathbf{c}) \cdot (\mathbf{e} + t\vec{\mathbf{d}} - \mathbf{c}) - r^2 = 0$$

Rearrange

$$(\vec{\mathbf{d}} \cdot \vec{\mathbf{d}})t^2 + 2\vec{\mathbf{d}} \cdot (\mathbf{e} - \mathbf{c})t + (\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - r^2 = 0$$

Looks familiar…

$$At^2 + Bt + C = 0$$



It’s a quadratic! (can use the quadratic equation)

Hint: the discriminant tells us what kinds of roots the equation has.

Intersection with a Triangle (Parametric Surface)

Check via equating point on surface with point on ray

$$\mathbf{p}(t) = \alpha \mathbf{t}_1 + \beta \mathbf{t}_2$$

$$\mathbf{e} + t\vec{\mathbf{d}} = \alpha \mathbf{t}_1 + \beta \mathbf{t}_2$$

$$\mathbf{e} = \alpha \mathbf{t}_1 + \beta \mathbf{t}_2 - t\vec{\mathbf{d}}$$

Intersection with a Triangle (Parametric Surface)

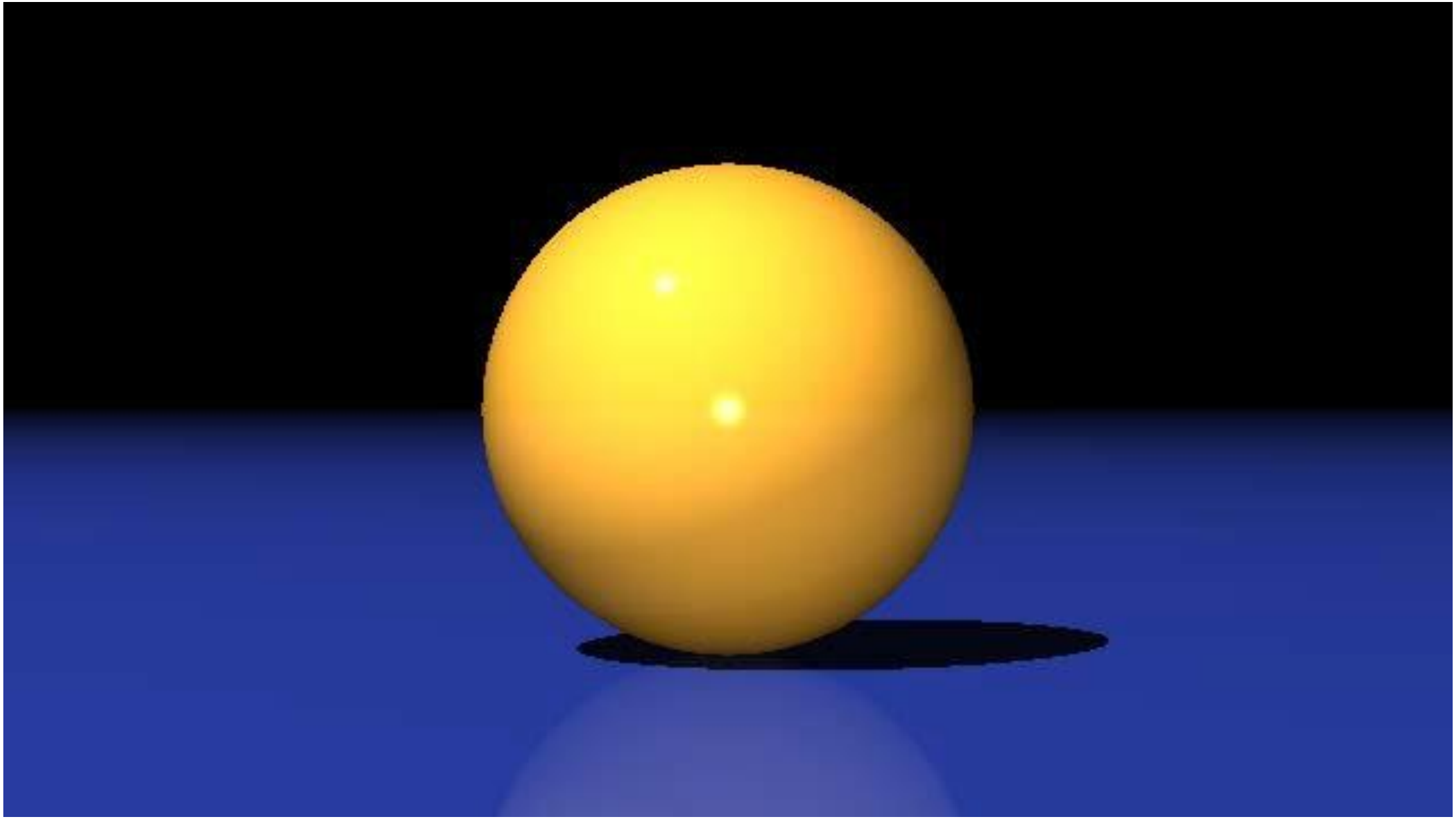
Check via equating point on surface with point on ray

$$\mathbf{e} = \alpha \mathbf{t}_1 + \beta \mathbf{t}_2 - t\vec{\mathbf{d}}$$

$$\mathbf{e} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & -\mathbf{d} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ t \end{bmatrix}$$

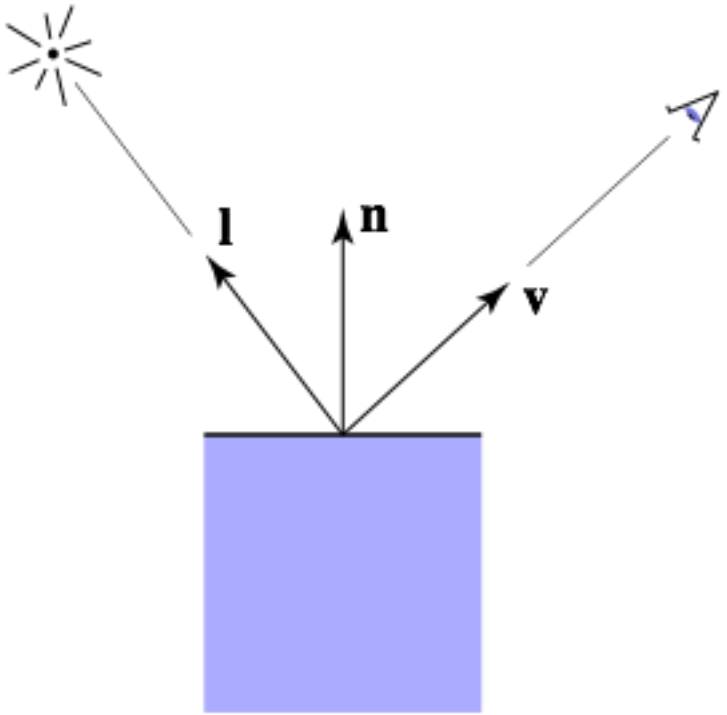
Check values of α, β, t

Ray Tracing



Shading

- Compute light reflected toward camera
- Inputs:
  - eye direction
  - light direction
    (for each of many lights)
  - surface normal
  - surface parameters
    (color, shininess, …)



Computing the Normal at a Hit Point

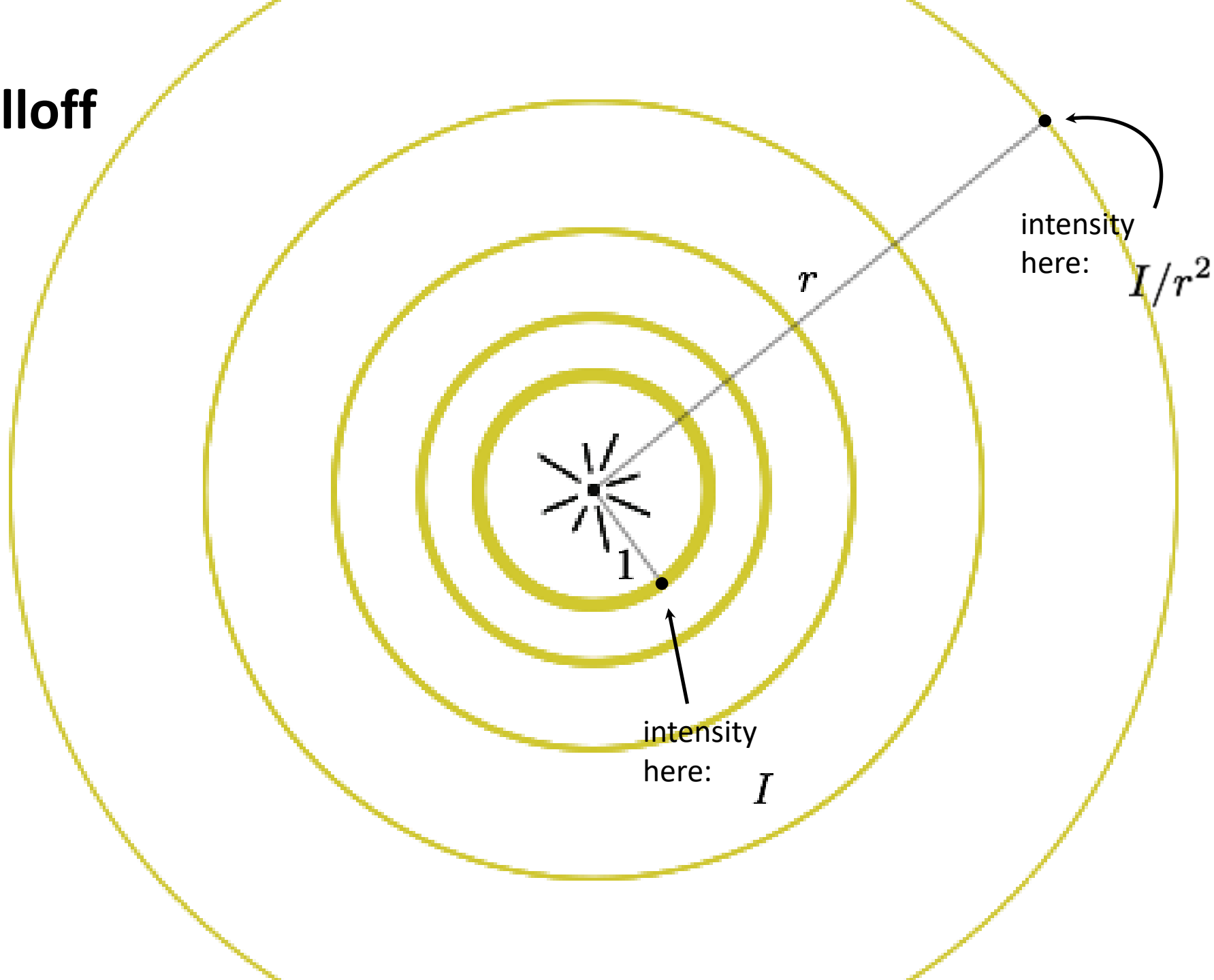
- Polygon normal: cross product of two non-collinear edges.
- Implicit surface normal $f(p)=0$:

$$gradient(f)(p).$$

- Explicit parametric surface $f(a,b)$:

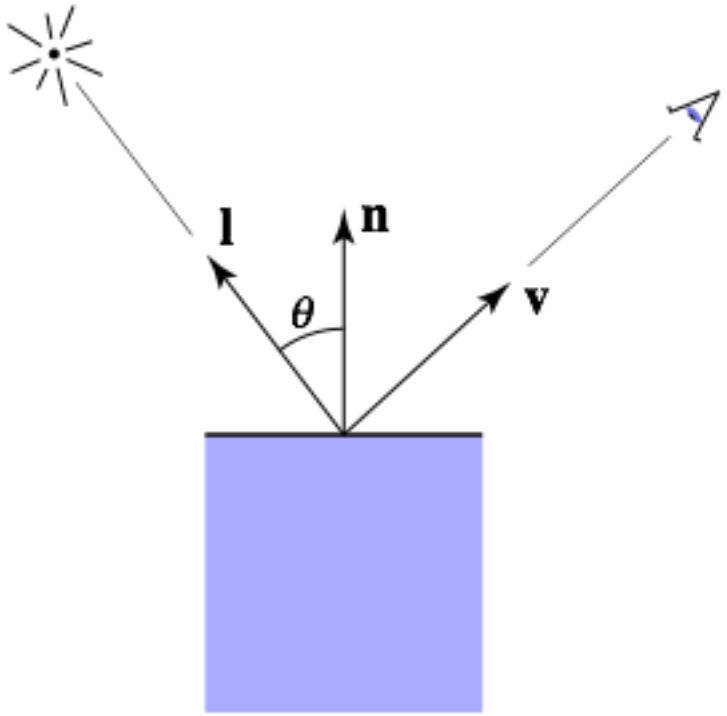
$$\delta f(s,b)/\ \delta s\ X\ \delta f(a,t)/\ \delta t.$$

Light falloff



Lambertian shading

Shading independent of view direction



illumination from source

$$L_d = k_d \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l})$$

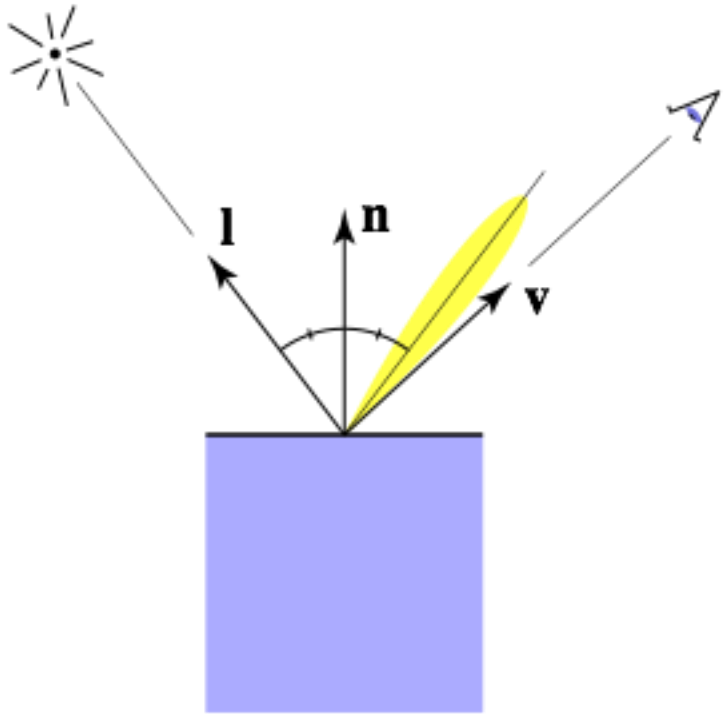
diffuse coefficient

diffusely reflected light

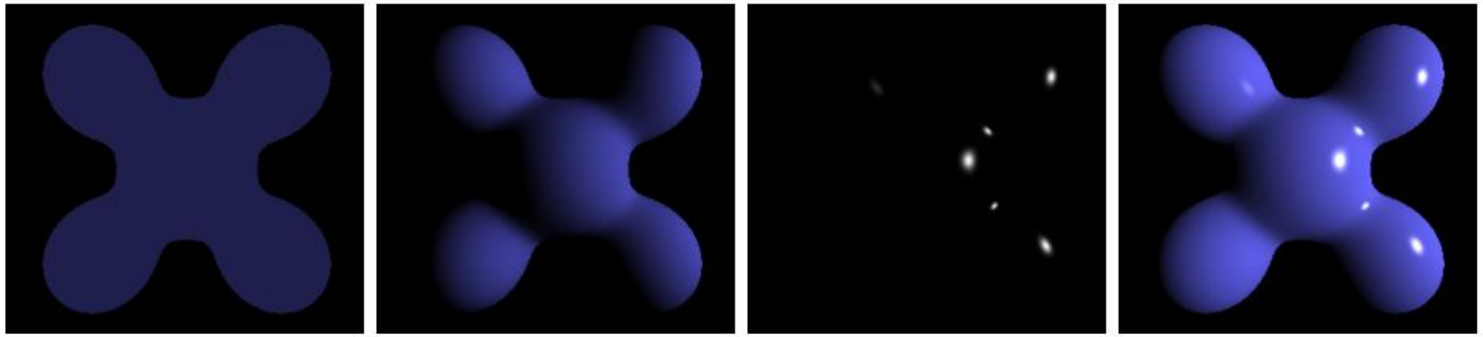
Specular shading (Phong)

Intensity depends on view direction

– bright near mirror configuration $k_s * I_s * (v.r)^{shiny}$



Local Illumination



- Usually include ambient, diffuse, Phong in one model

$$L = L_a + L_d + L_s$$
$$= k_a\, I_a + k_d\, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s\, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

- The final result is the sum over many lights

$$L = L_a + \sum_{i=1}^{N} [(L_d)_i + (L_s)_i]$$

$$L = k_a\, I_a + \sum_{i=1}^{N} \left[k_d\, (I_i/r_i^2) \max(0, \mathbf{n} \cdot \mathbf{l}_i) + k_s\, (I_i/r_i^2) \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p\right]$$

Local vs. Global Illumination

Local Illumination Models

- e.g. Phong, Blinn.
- Model source from a light reflected once off a surface towards the eye.
- Indirect light is included with an ad hoc “ambient” term which is normally constant across the scene.

Global Illumination Models

- e.g. recursive ray tracing (incomplete model).
- Try to measure light propagation in the scene.
- Model interaction between objects, other objects, and their environment

Path Tracing

- A ray from a light L can bounce of any number of specular S and diffuse objects D before entering the eye E. The paths from E to L for eg. can be

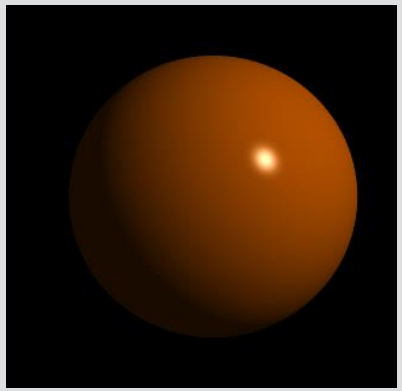
  E-D-L, E-S-D-D-S-S-D-S-L, E-D-D-S-D-S-L…

- Rays are infinitely thin
- Don’t disperse

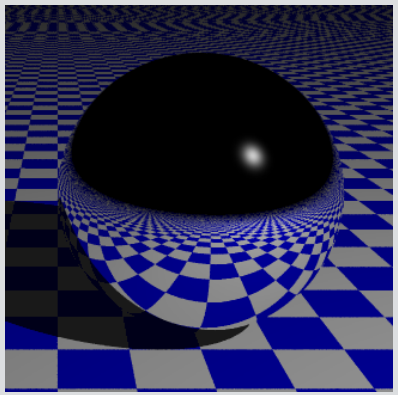
- Ray Tracing model shiny objects exhibiting multiple reflections, i.e. paths of the form

  $E - S^{*} - D^{+} – L.$

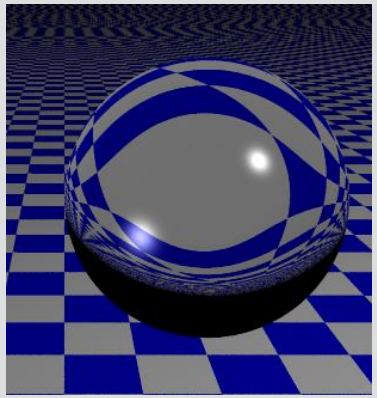
local illumination

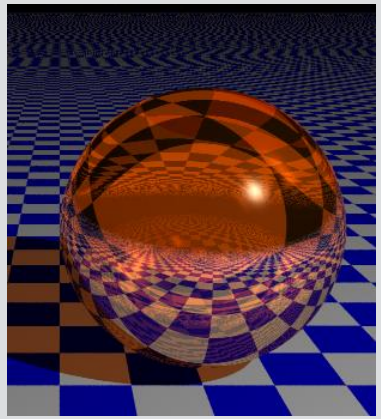


reflection



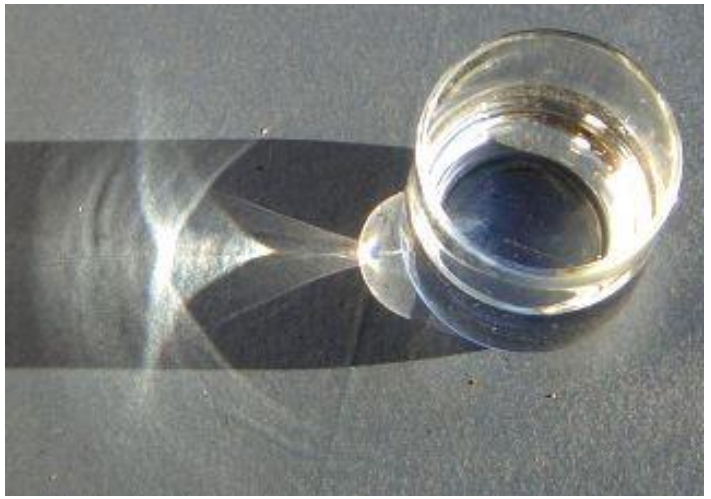
refraction

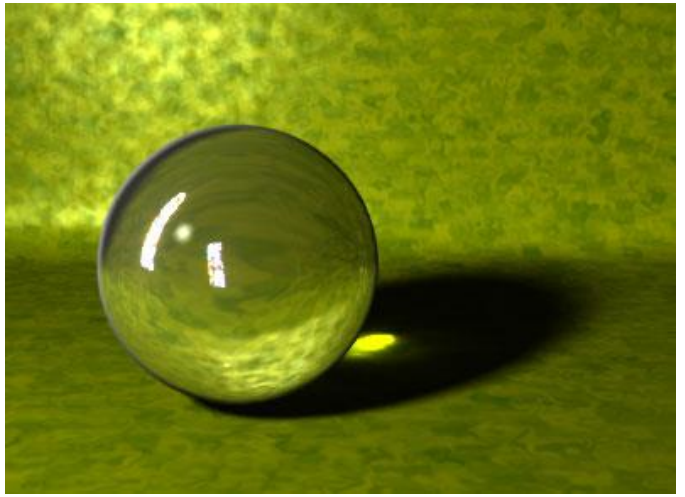




Ray Tracing Improvements: Caustics

- Transport E – S – S – S - D – S – S – S - L
- Trace from the light to the surfaces and then from the eye to the surfaces
- “shower” scene with light and then collect it
- “Where does light go?” vs “Where does light come from?”
- Good for caustics





The Rendering Equation

outgoing light at position **x** and direction **w**

emitted light at position **x** and direction **w**

and

reflected light at position **x** and direction **w**

the reflected light at position **x** and direction **w**

is the integral over all possible directions **w'**

the incoming light from all directions

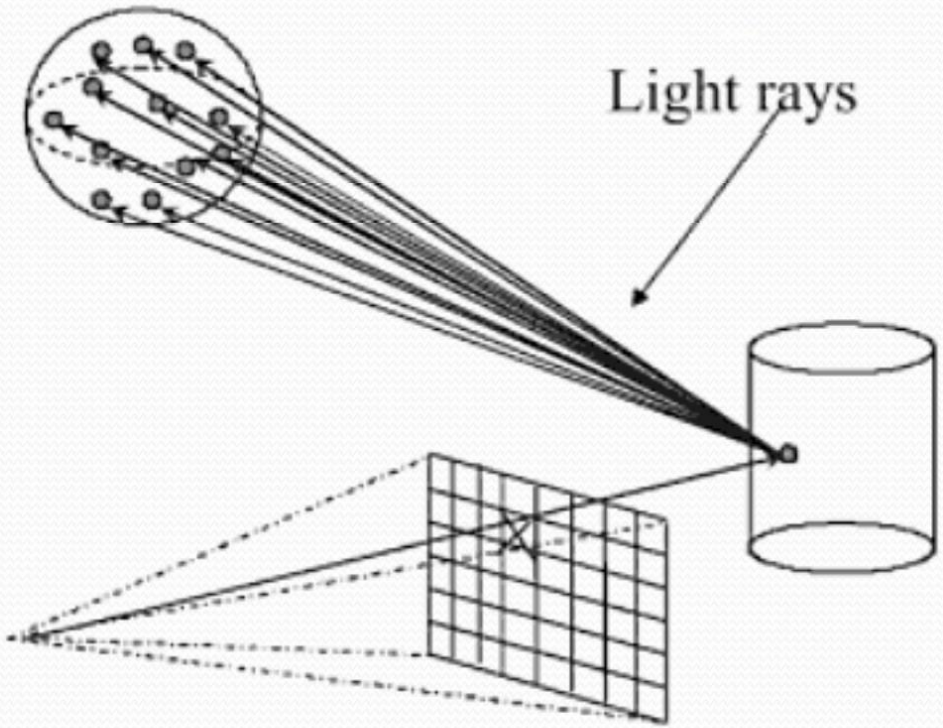
times

BRDF:
a function describing how light is reflected at an opaque surface

Area Light

Monte-Carlo Area light

- Light is modeled as a sphere
- Highest intensity in the middle. Gradually fade out.
- Shoot n rays to random points in the sphere
- Average their value.

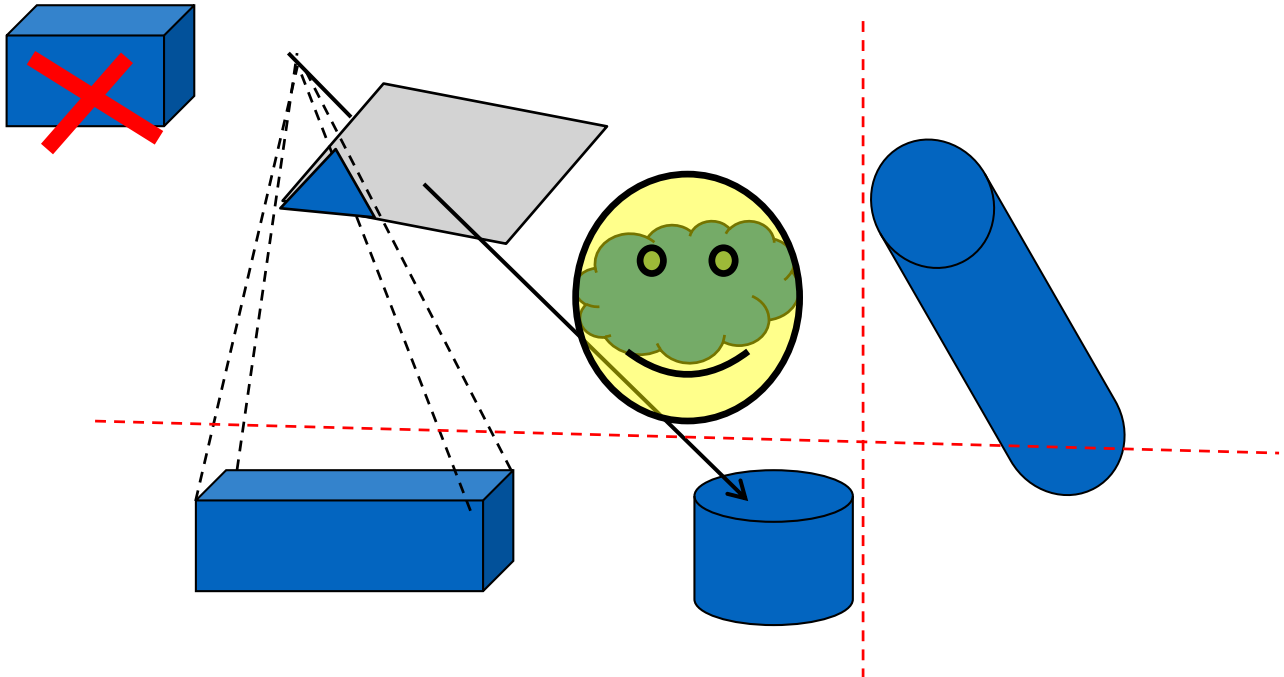


http://courses.cs.tau.ac.il/Computer-Graphics/10a/lectures/05a-RayTracingAdv.pdf

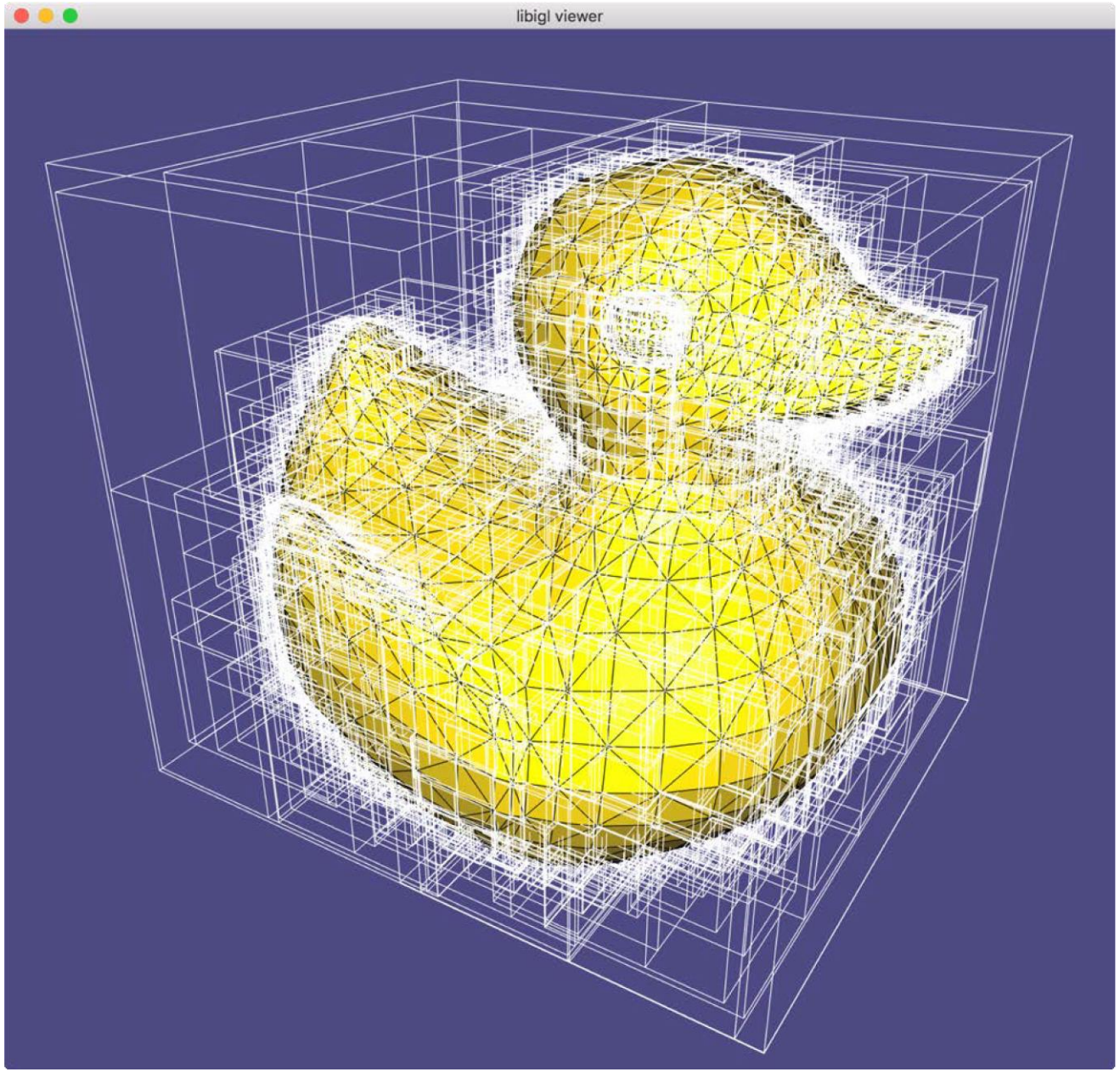
Ray Intersection: Efficiency Considerations

Speed-up the intersection process.

- Ignore object that clearly don’t intersect.
- Use proxy geometry.
- Subdivide and structure space hierarchically.
- Project volume onto image to ignore entire sets of rays.



Boundary Volume Hierarchies



Bounding volumes

Quick way to avoid expensive testing intersections and collisions:

bound object with a simple volume **bvol** (volume encloses object)

If a ray/object doesn't hit/intersect **bvol**,

it doesn't hit/intersect the object

else

test object for hit/intersect

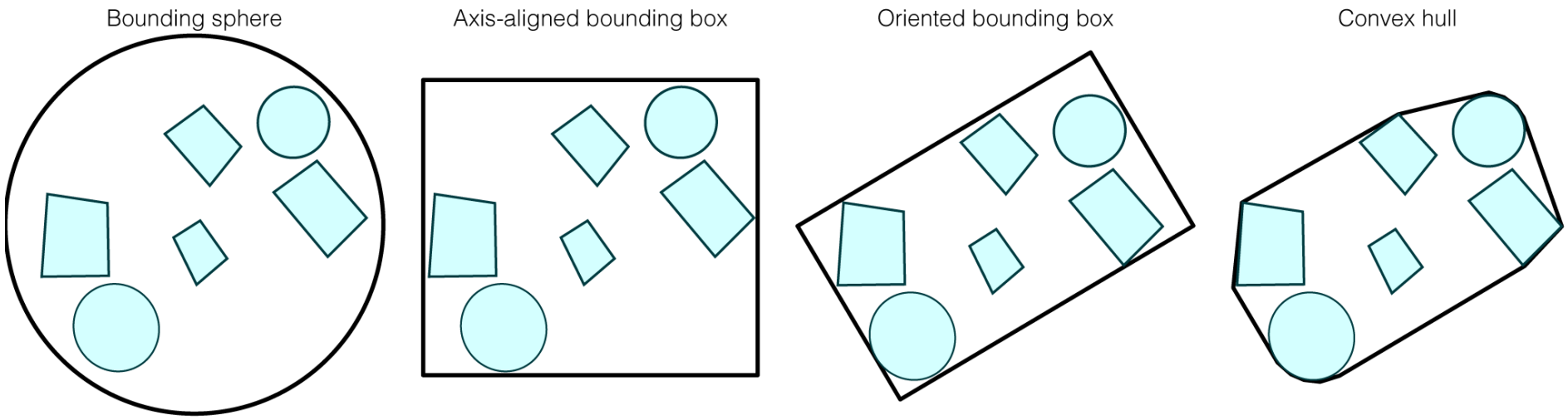
Cost: more for hits and near misses, less for far misses

Worth doing? Yes if:

- Cost of **bvol** intersection test is small: simple shapes (spheres, boxes, …)
- Cost of object intersect test is large: **bvol** most useful for complex objects
- Tightness of fit is good:
  - Loose fit leads to extra object intersections
  - Tradeoff between tightness and **bvol** intersection cost

Choice of bounding volumes

Spheres: easy to intersect, not always tight.
Axis-aligned Bounding Boxes (AABBs): easy to intersect, tighter for axis-aligned objects.
Oriented bounding boxes (OBBs): easy to intersect (transformation cost),
tighter than AABBs.
Convex Hull: not as easy to intersect as the above, tighter than the above.

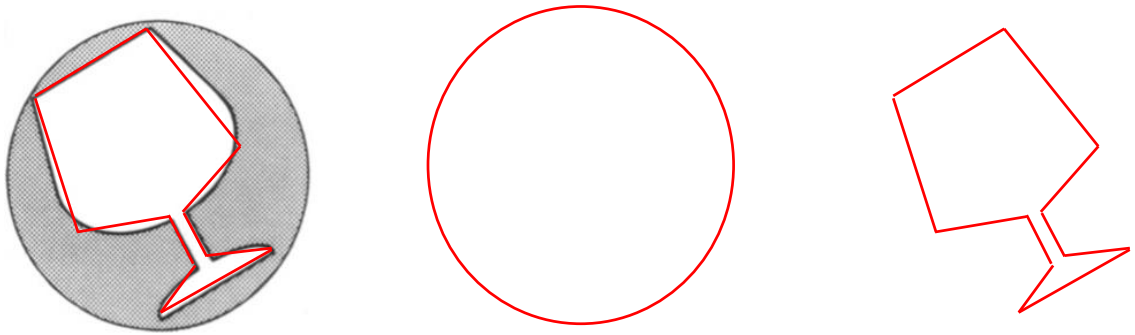


Proxy Geometry vs. Bounding volumes

Another concept often used in CG is proxy geometry or Level-Of-Detail LOD.

A proxy is a simplified representation of the object, that can be used as the object when rendering and processing speed is more important than visual accuracy.

Note, that proxy geometry is an approximation to the object and typically not a bounding volume. And a bounding volume itself is typically not a good visual proxy for an object.



[Glassner 89, Fig 4.5]

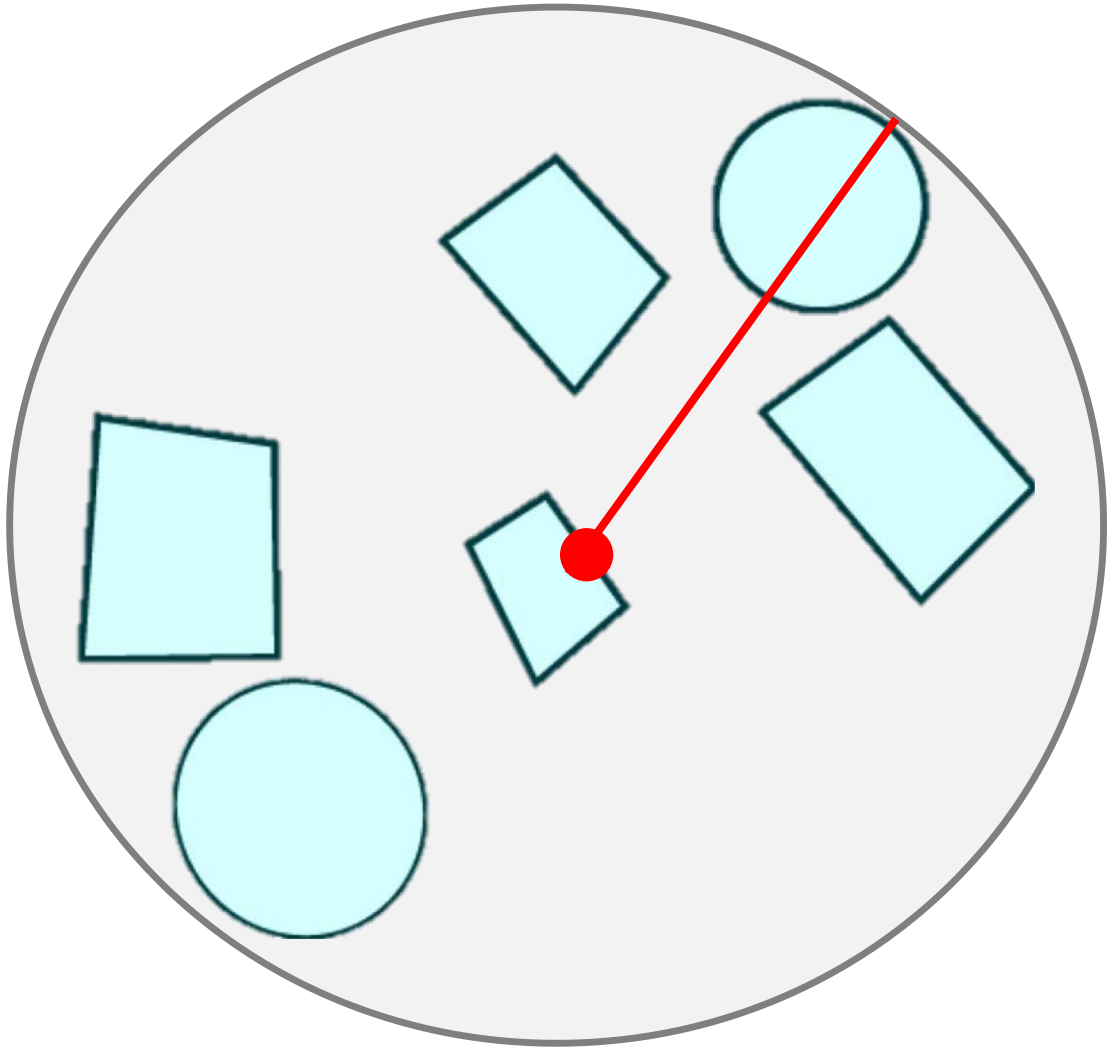
Building a Bounding Sphere

Parameters of a Sphere:

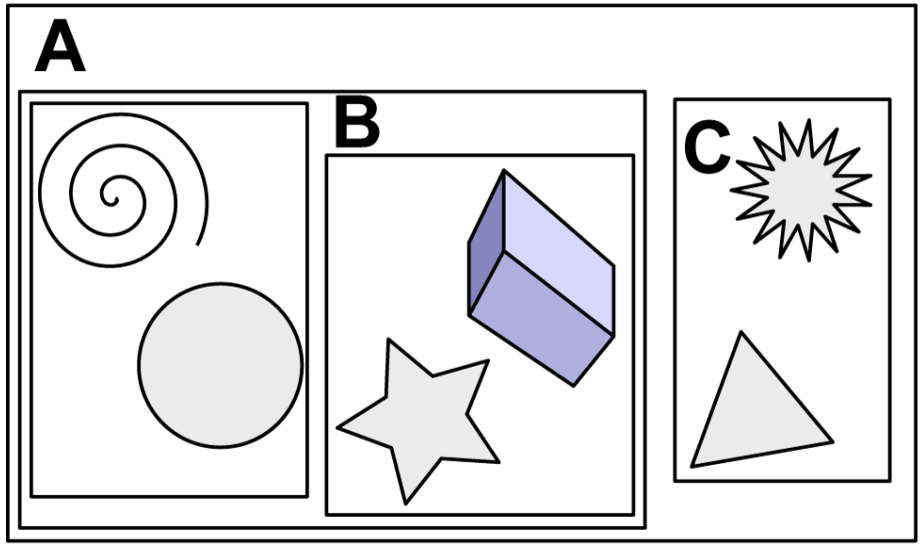
1. Center = $\mathbf{c} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{v}^i$

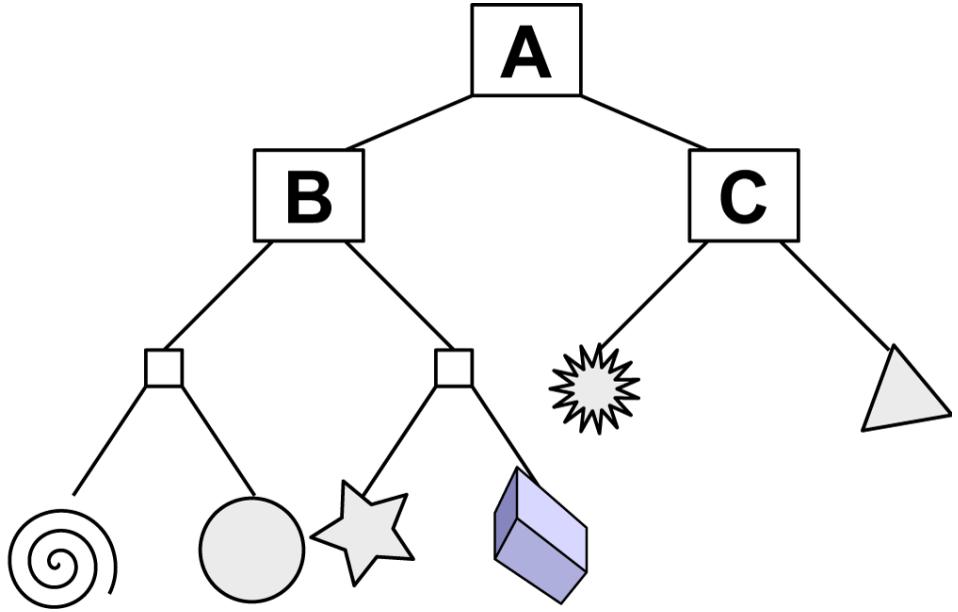
2. Radius = $r = \max(\mathbf{v}^i - \mathbf{c})$

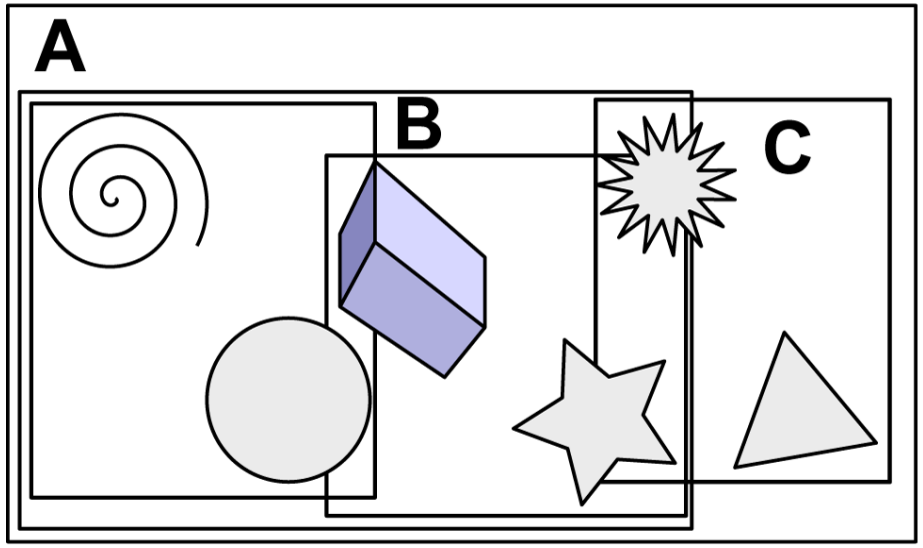
$$\mathbf{v}^i \in \text{Vertices}$$

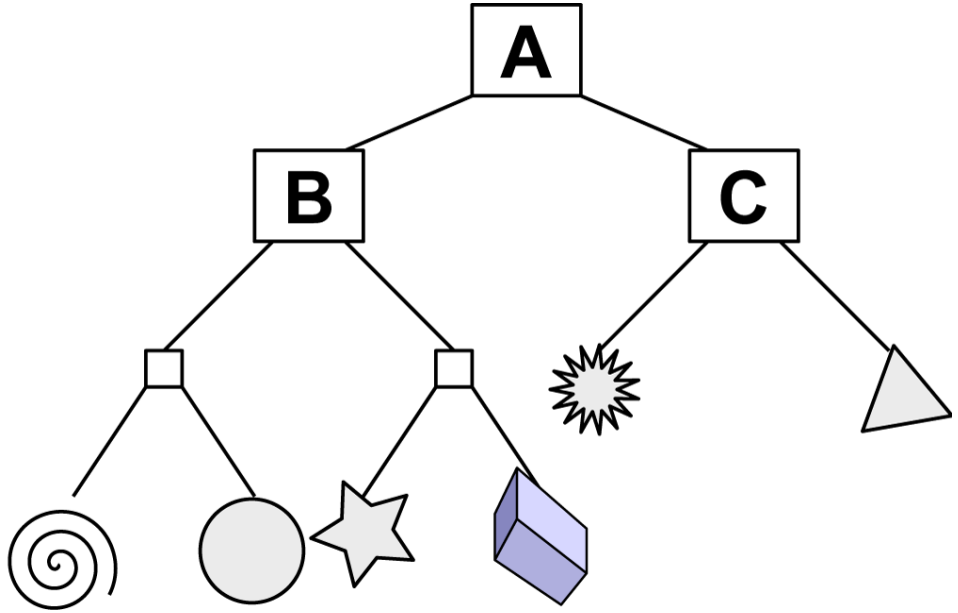


AABB Tree Construction (object based)



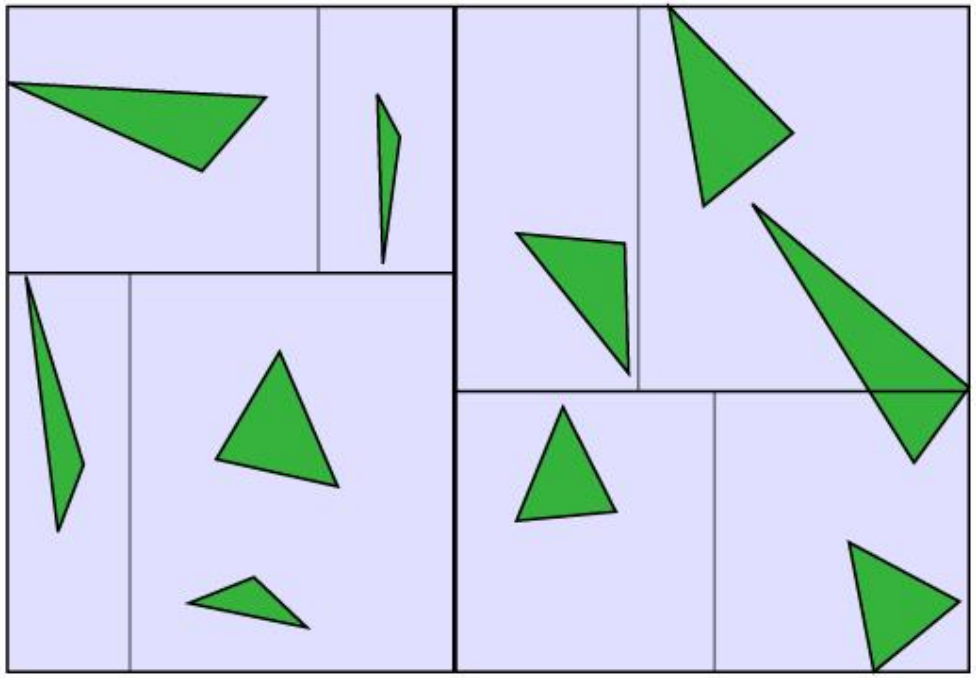


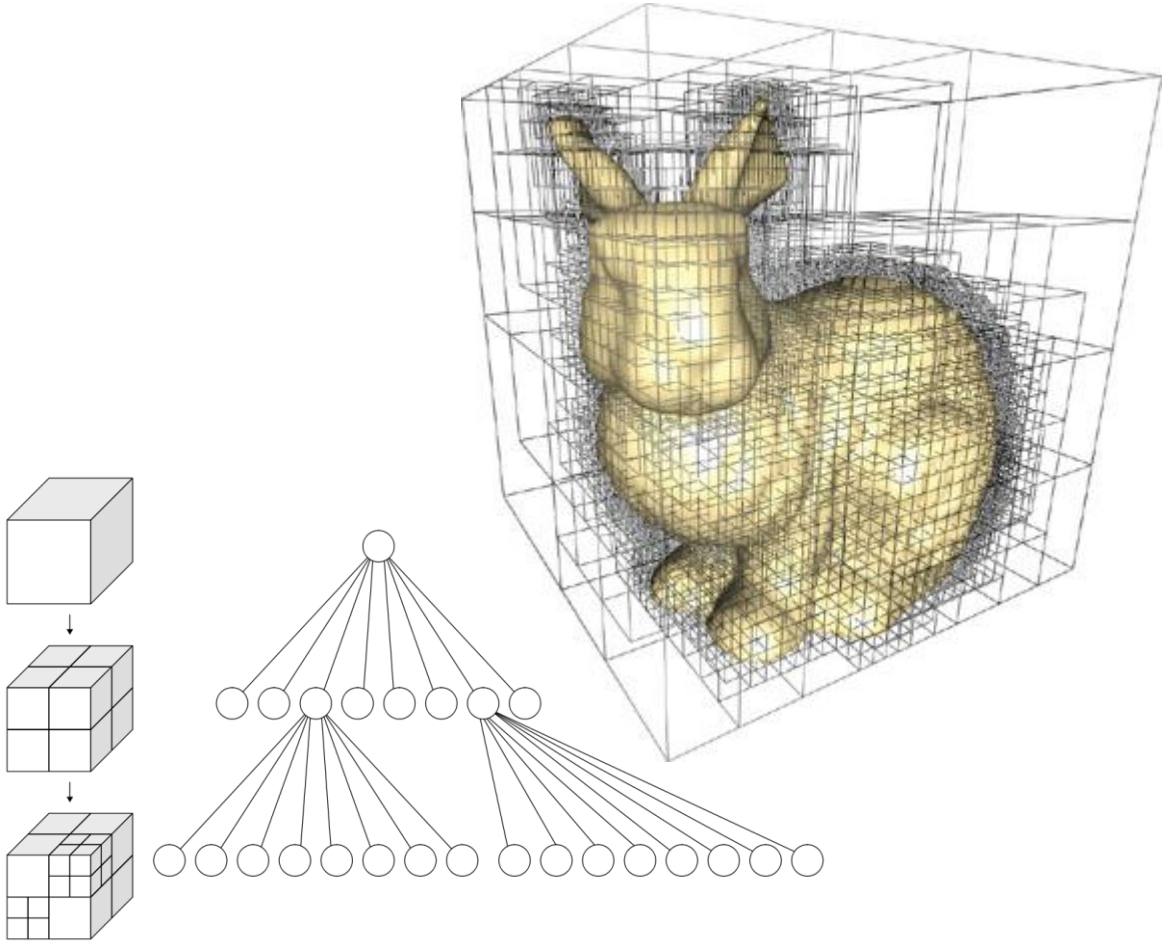




Non-regular space subdivision

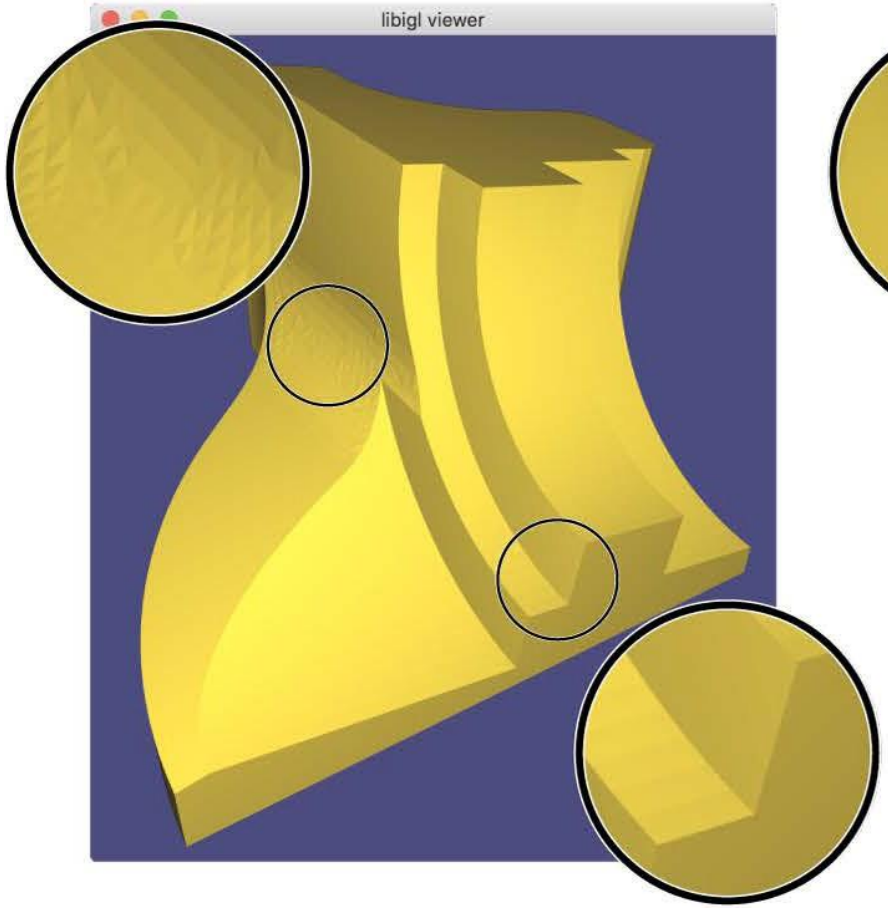
k-d Tree
Octree



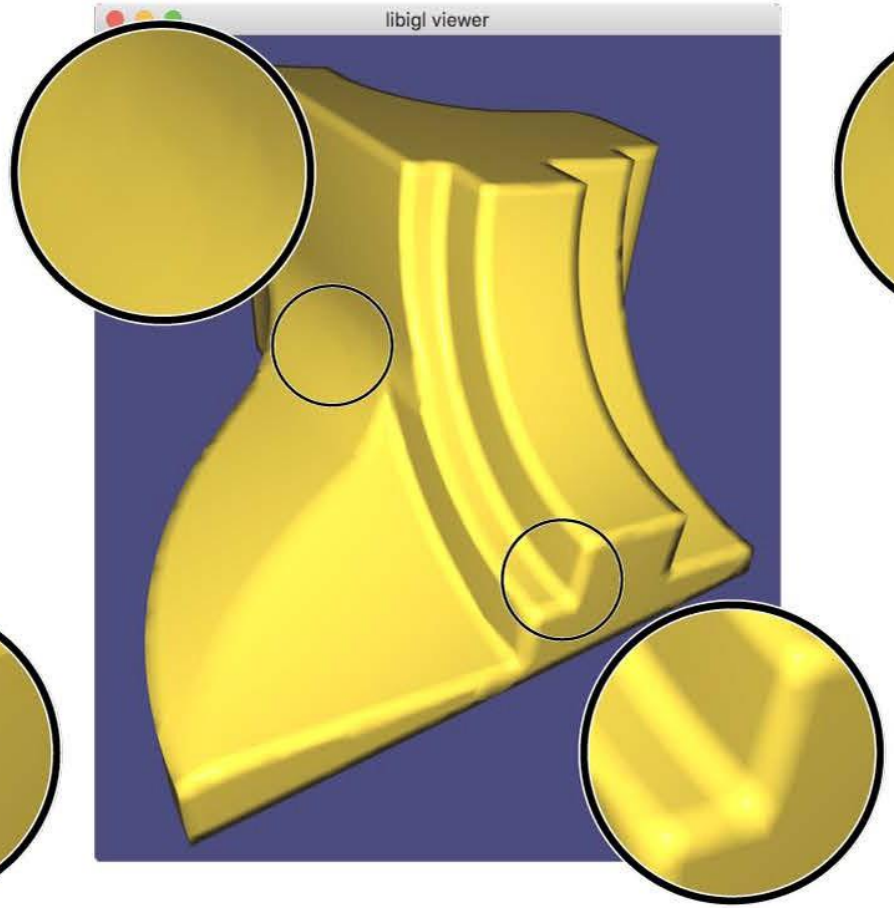


Meshes

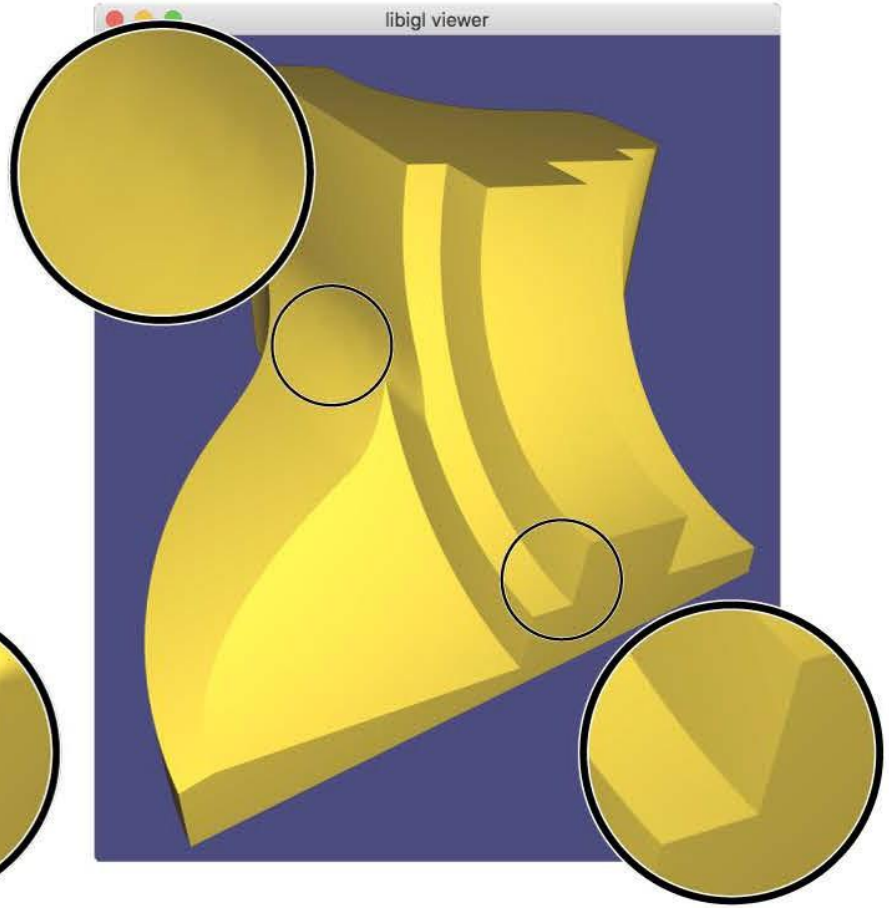
Per-face normals

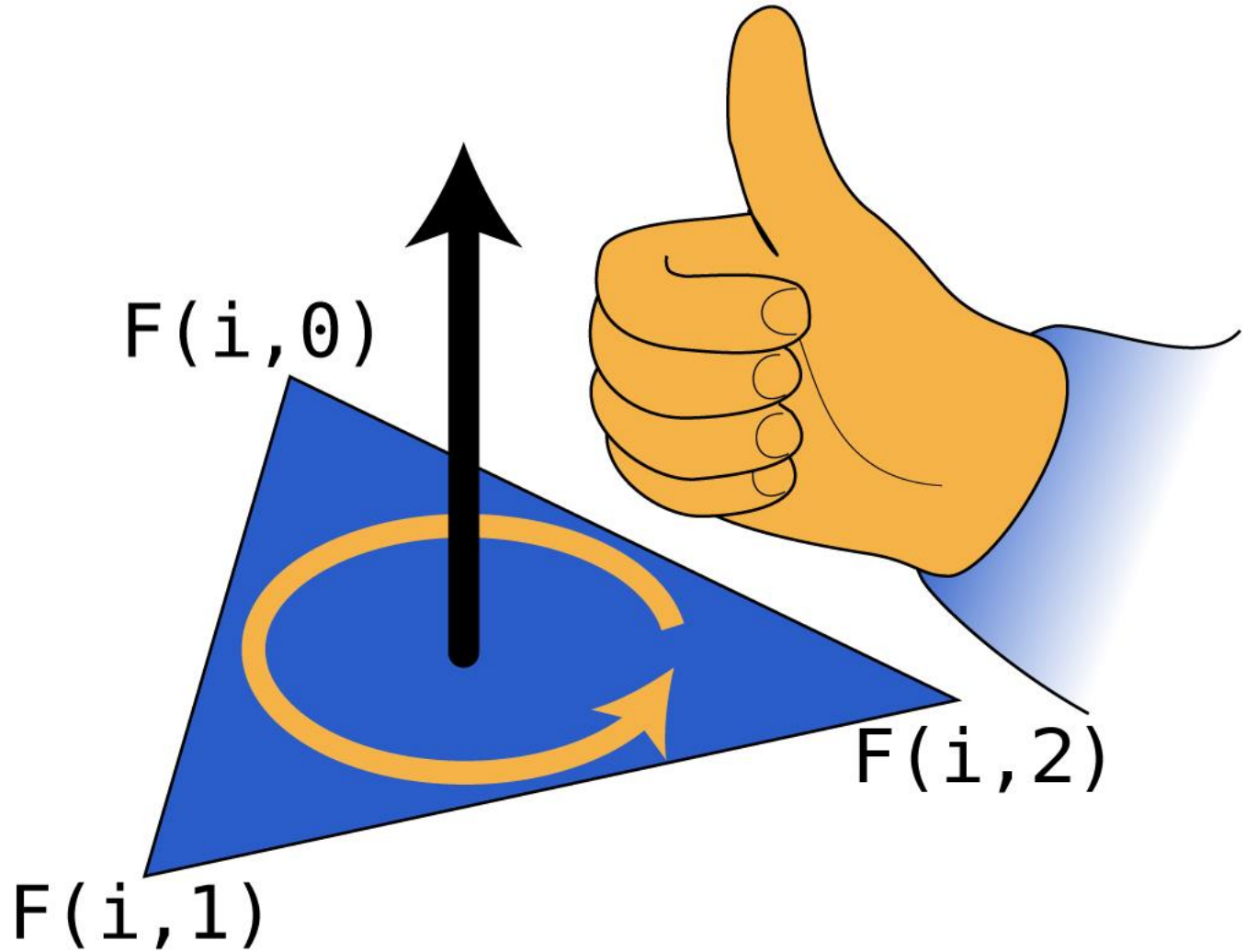


Per-vertex normals



Per-corner normals

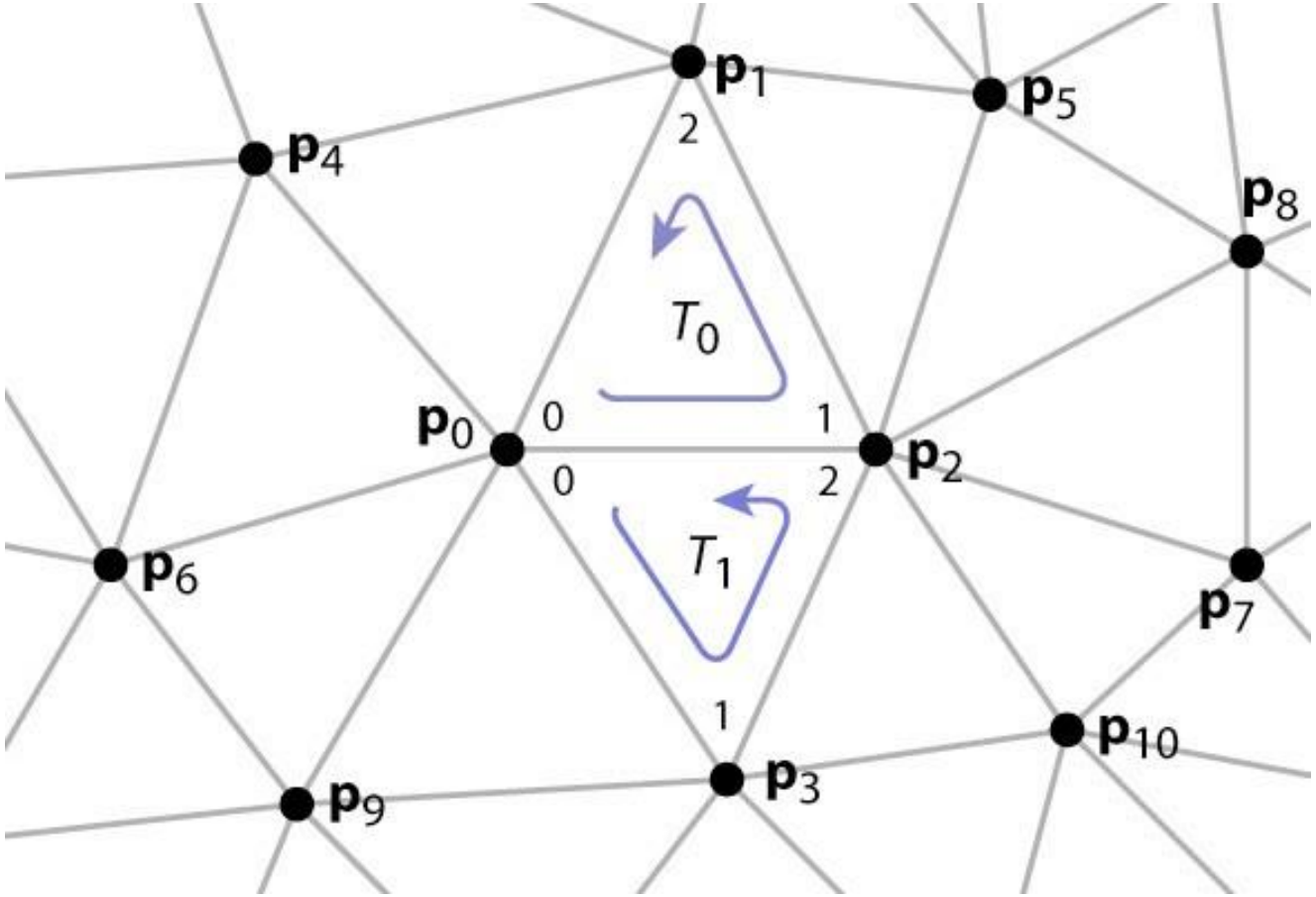



F(i,0)
F(i,2)
F(i,1)

Indexed triangle set

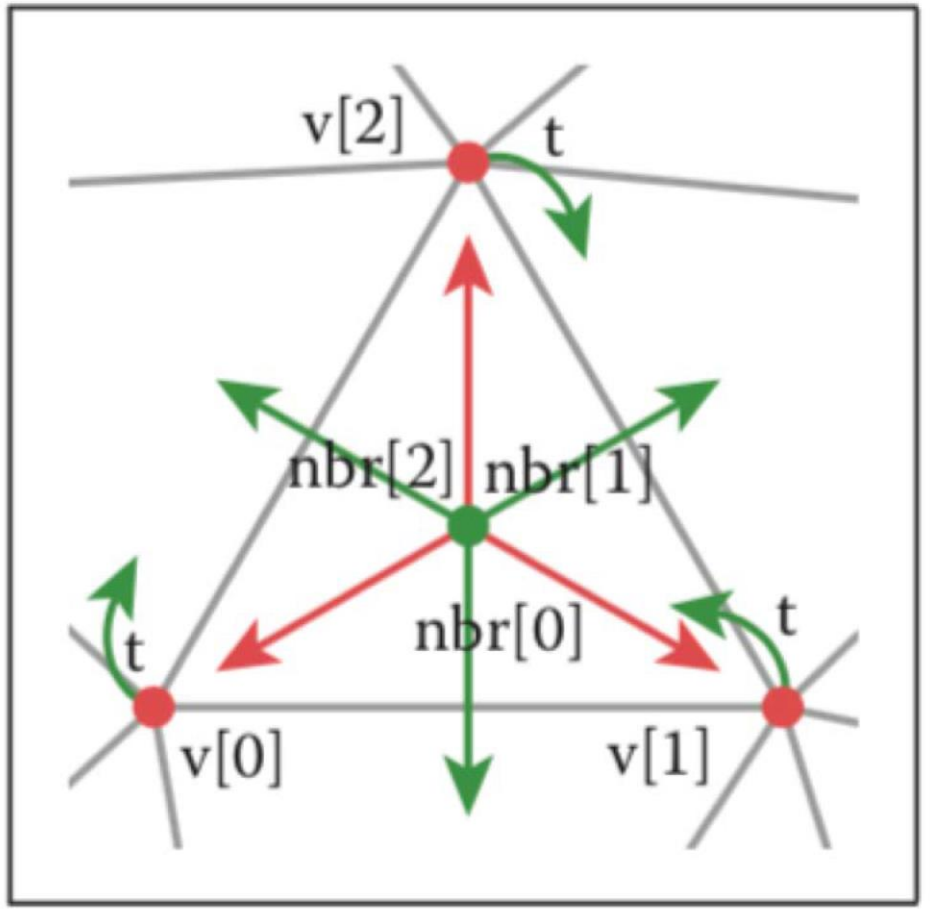
verts[0]	x_0, y_0, z_0
verts[1]	x_1, y_1, z_1
	x_2, y_2, z_2
	x_3, y_3, z_3
	⋮

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
	⋮



Triangle-Neighbour Data Structure

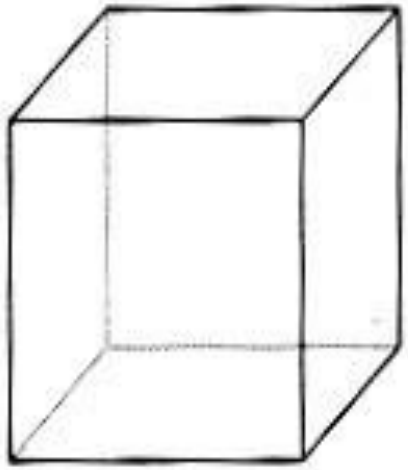
```
Mesh {
        float3 verts[nv];     // 3D vertex positions
        int vTri[nv];         // index of any adjacent triangle

        int tInd[nt][3];      // vertex indices
        int tNbr[nt][3];      // indices of neighbor triangles
        }
```



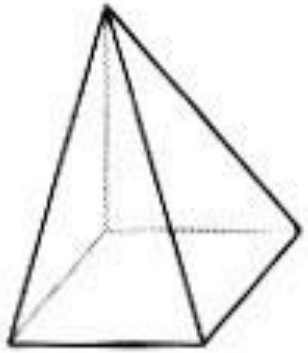
Relationships between primitive Types

What is the relationship between the number of vertices, the number of edges and the number of triangles in a mesh?

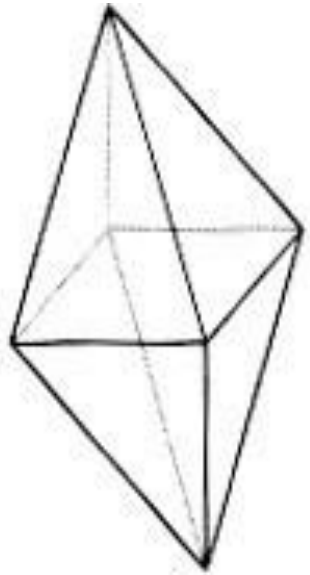
Euler's Formula: V+F-E = 2 -2g
(closed manifold mesh)



V = 8
E = 12
F = 6



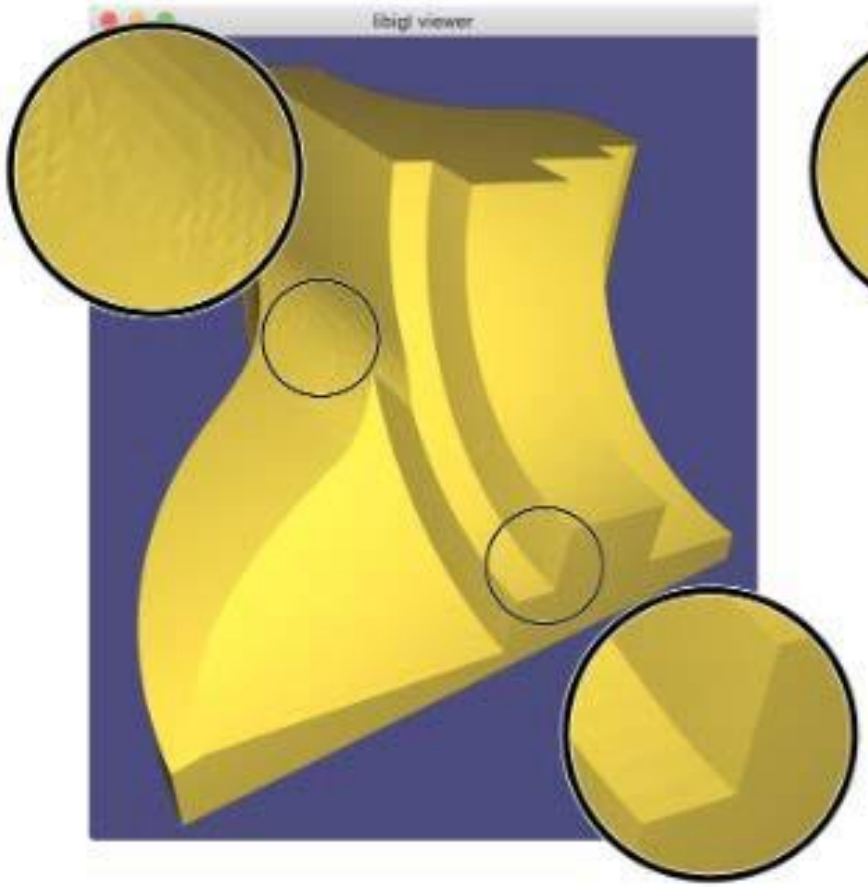
V = 5
E = 8
F = 5



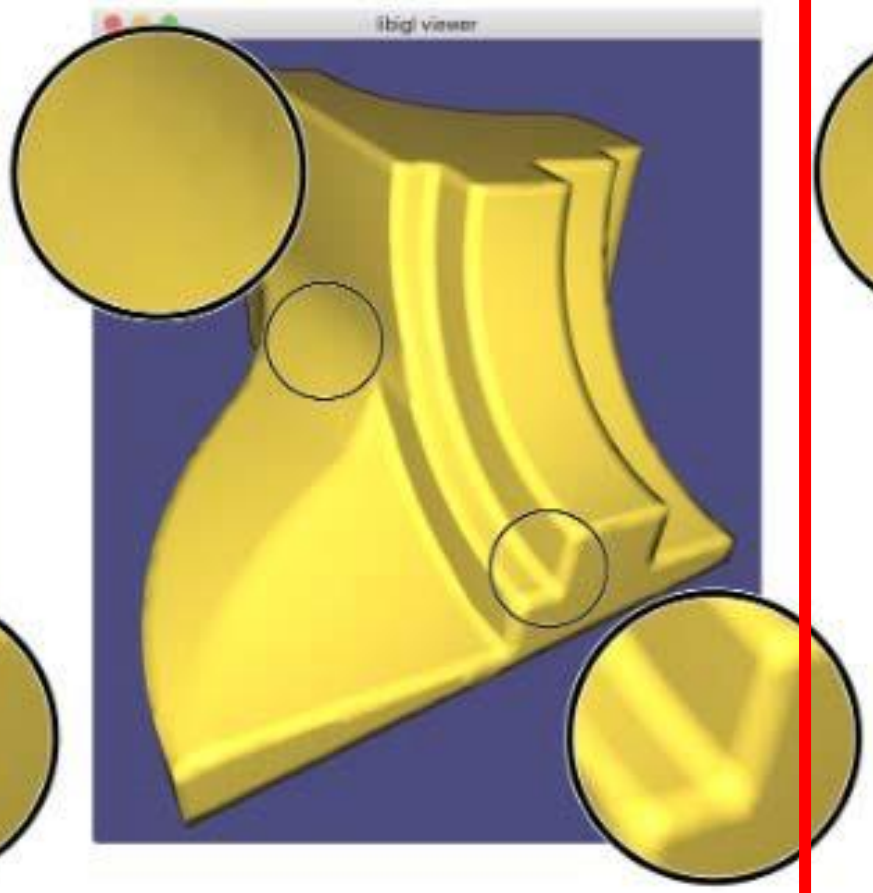
V = 6
E = 12
F = 8

Per-Corner Normals

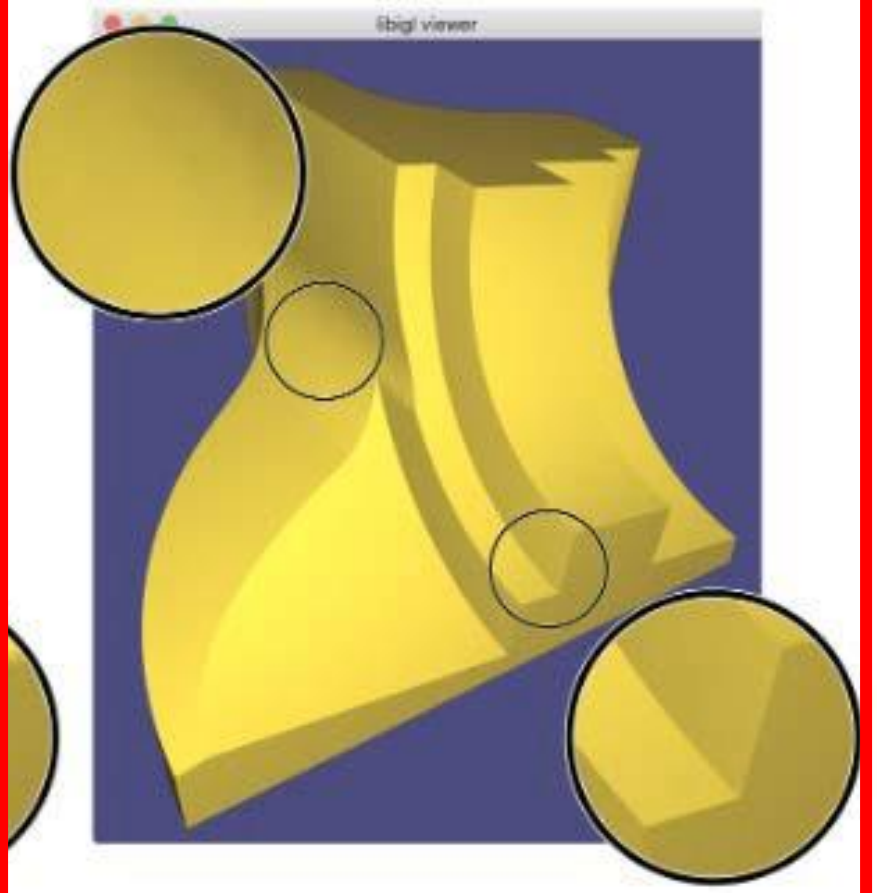
Per-Face Normals



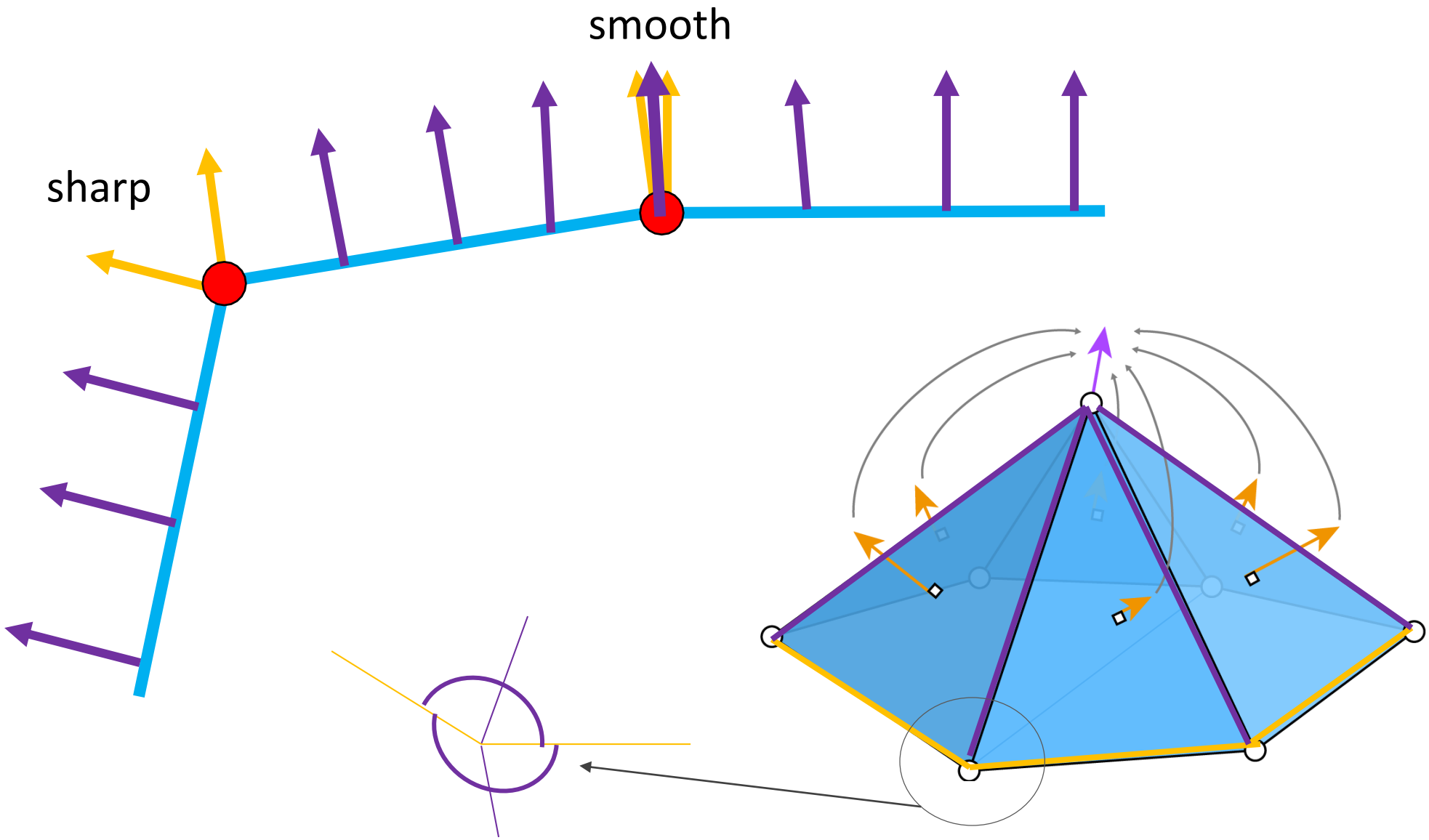
Per-Vertex Normals



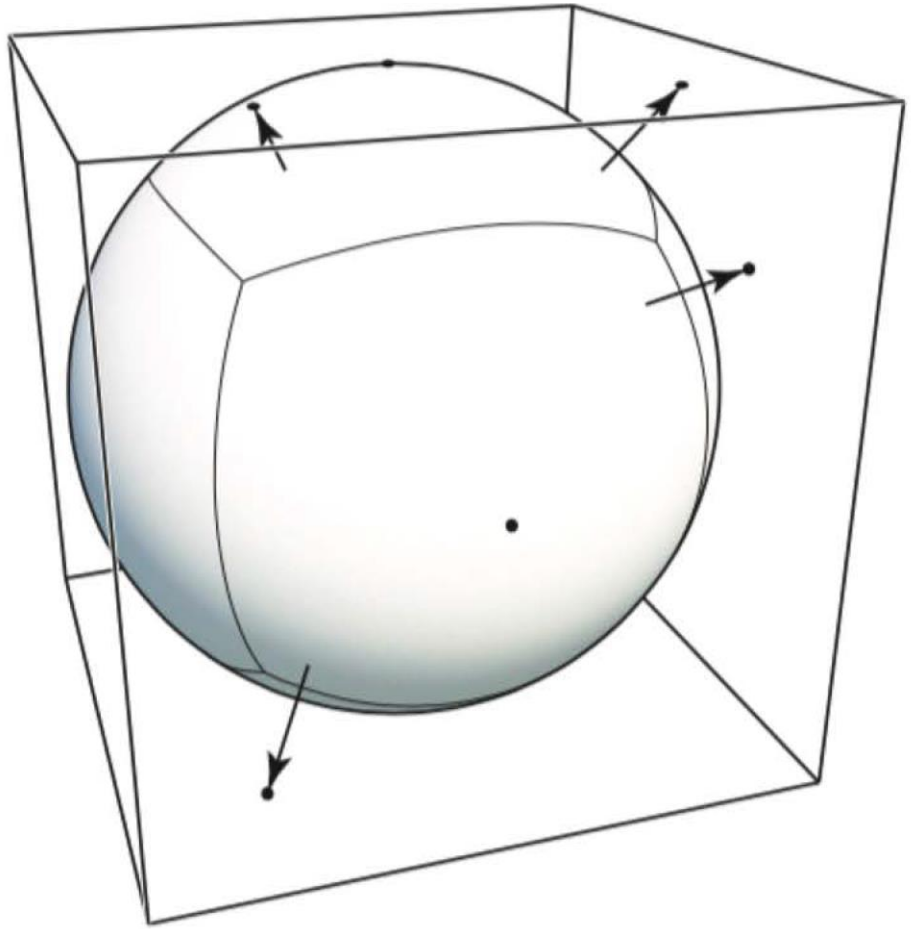
Per-Corner Normals

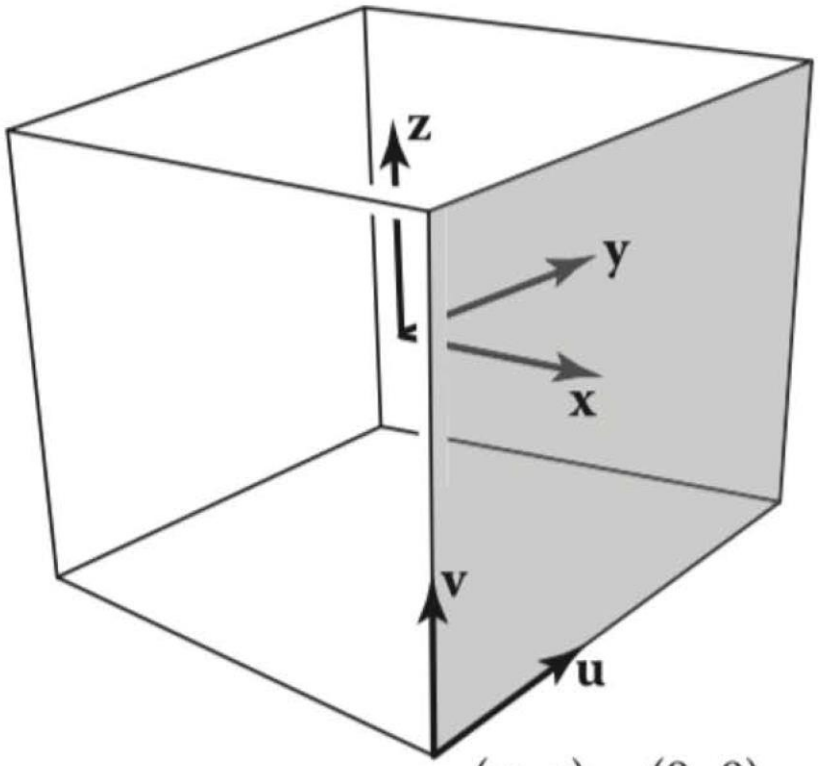


Per-Corner Normals



Cube Texture Map





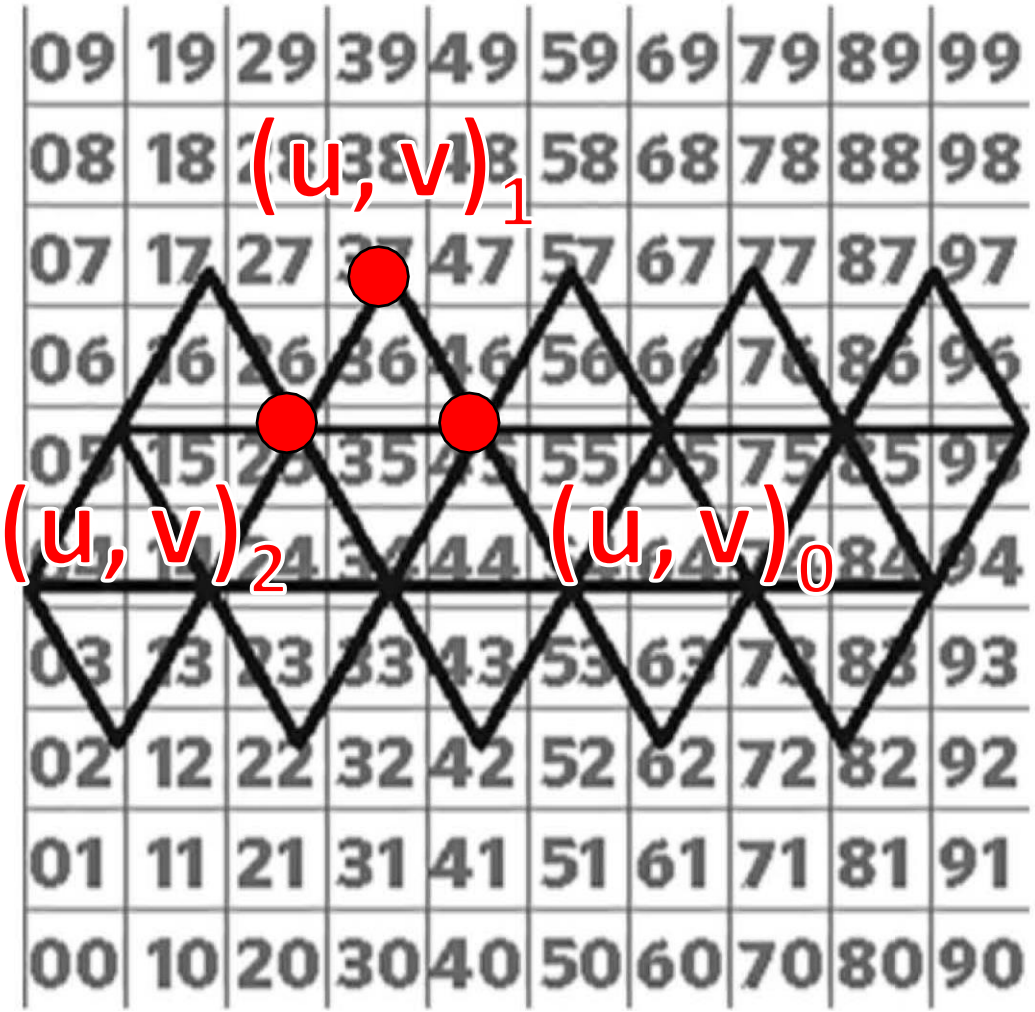
$(u, v) = (1, 1)$
$x = y = z$

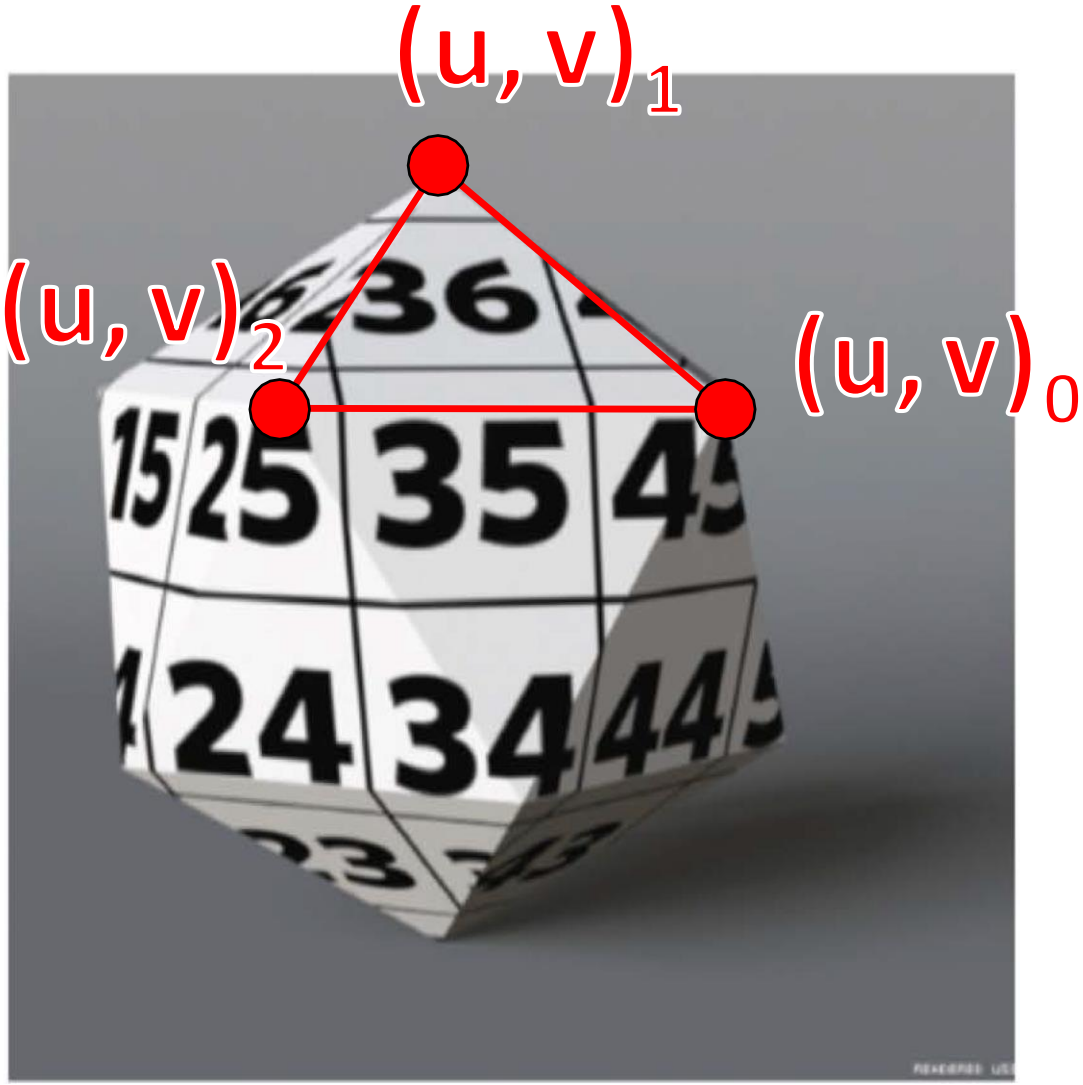
Right face has
$x > |y|$ and
$x > |z|$

$(u, v) = (0, 0)$
$x = -y = -z$

$$(x, y, z) \mapsto \left(\frac{x}{z}, \frac{y}{z}\right).$$

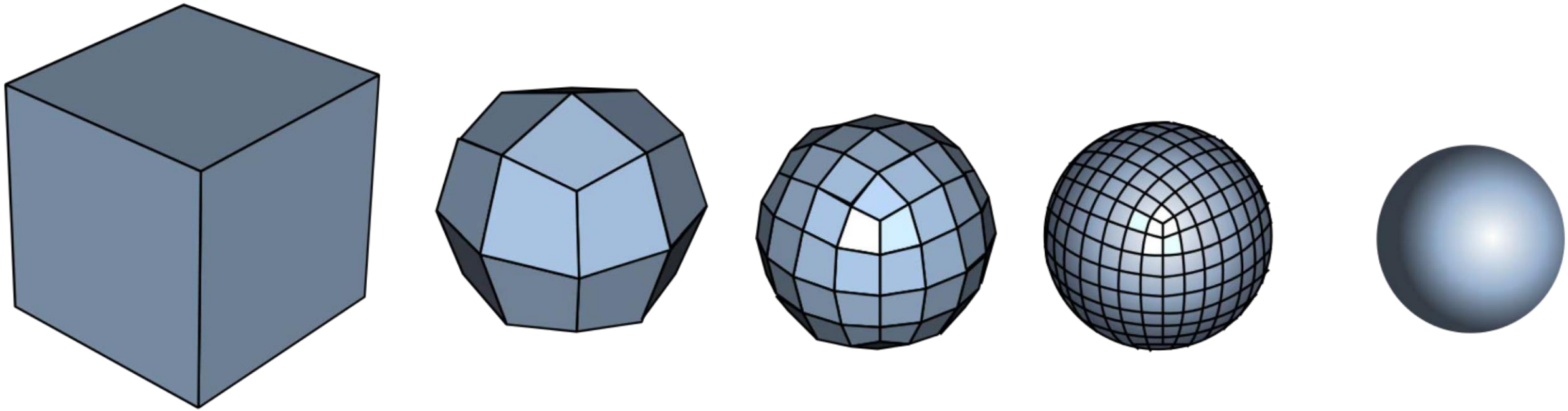
Interpolated Texture Coordinates





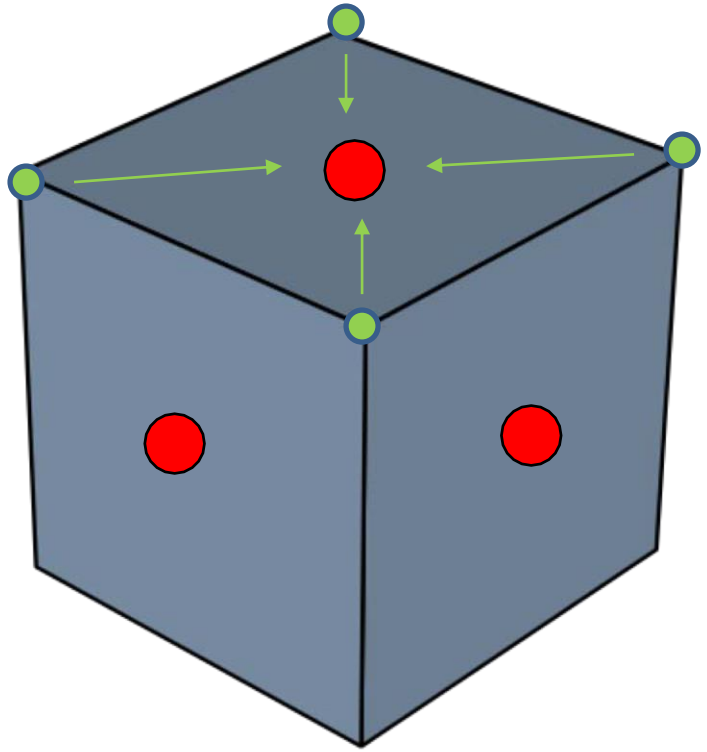
Catmull-Clark Subdivision

Particular type of subdivision scheme.



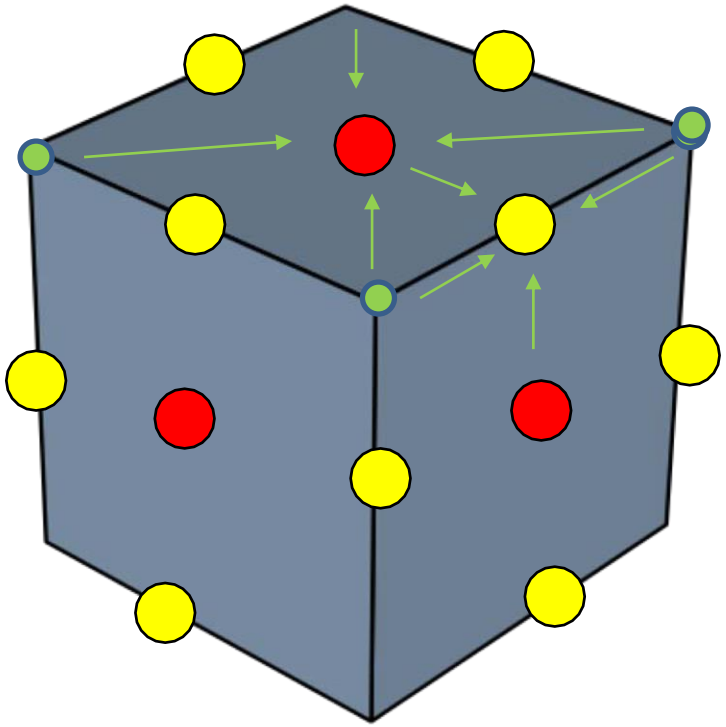
Catmull-Clark Subdivision

Step 1: Set the face point for each facet to be the average of its vertices



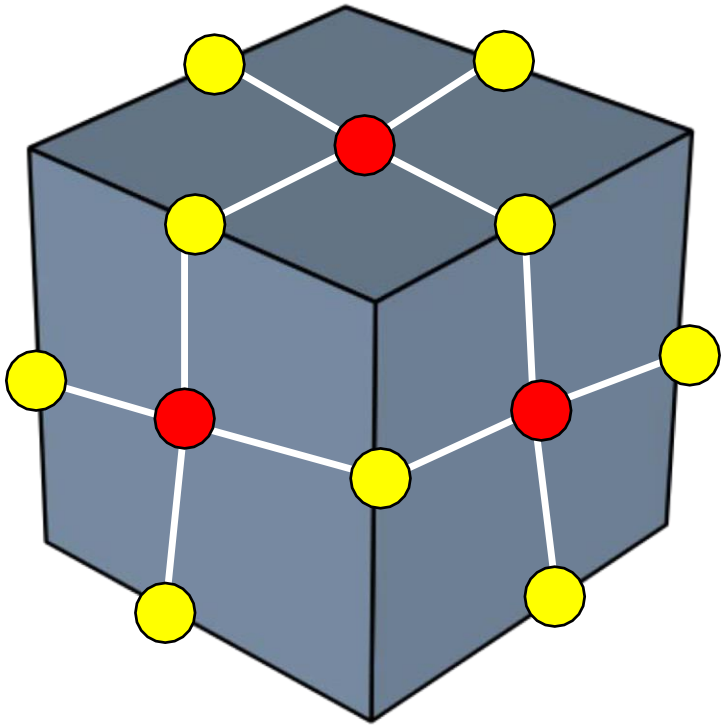
Catmull-Clark Subdivision

Step 2: Add edge points – average of two neighbouring face points and edge end points



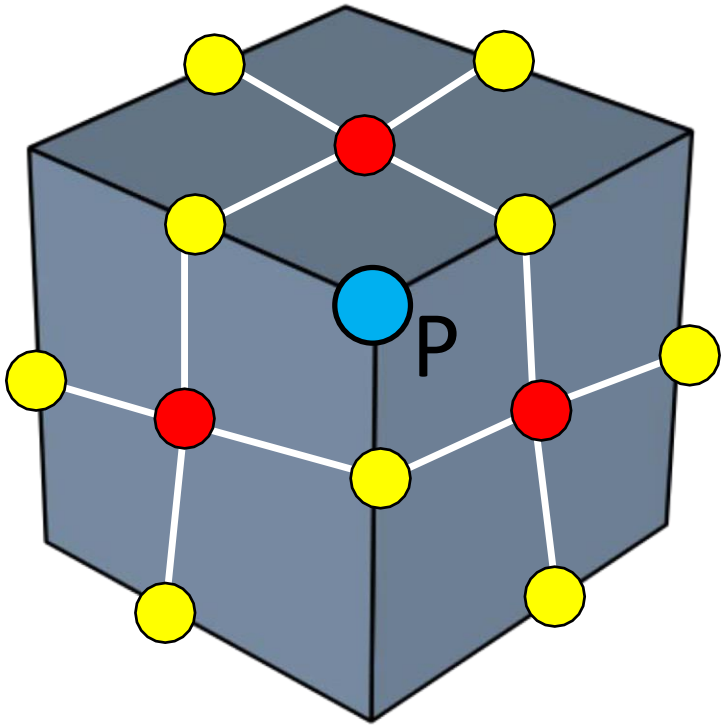
Catmull-Clark Subdivision

Step 3: Add edges between face points and edge points



Catmull-Clark Subdivision

Step 4: Move each original vertex according to new position given by:



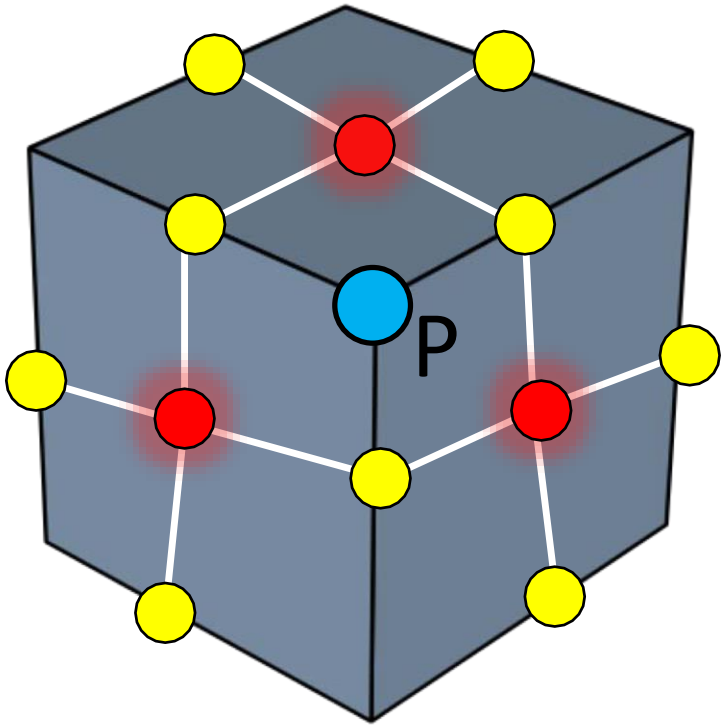
$$\frac{F + 2R + (n-3)P}{n}$$

F : Average of all n created face points adjacent to P

R : Average of all original edge midpoints touching P

Catmull-Clark Subdivision

Step 4: Move each original vertex according to new position given by:



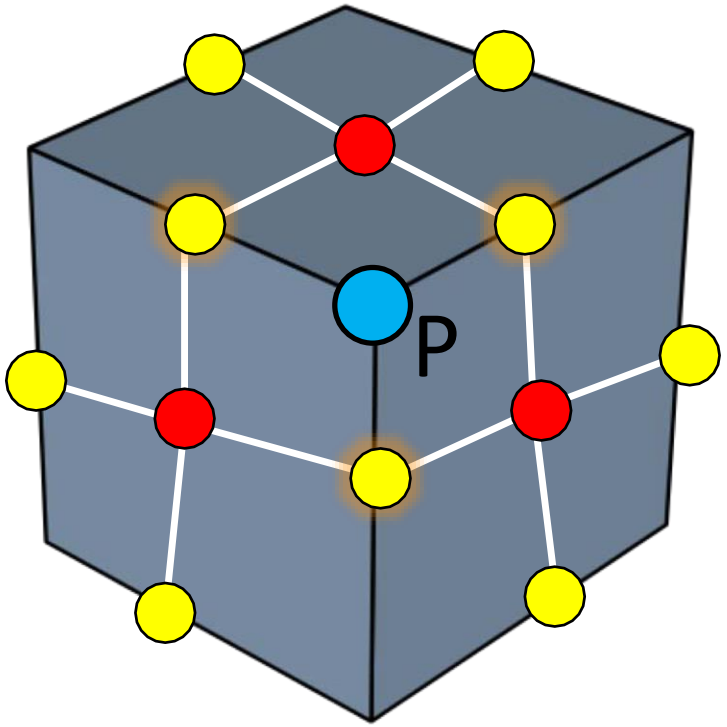
$$\frac{F + 2R + (n-3)P}{n}$$

F : Average of all n created face points adjacent to P

R : Average of all original edge midpoints touching P

Catmull-Clark Subdivision

Step 4: Move each original vertex according to new position given by:



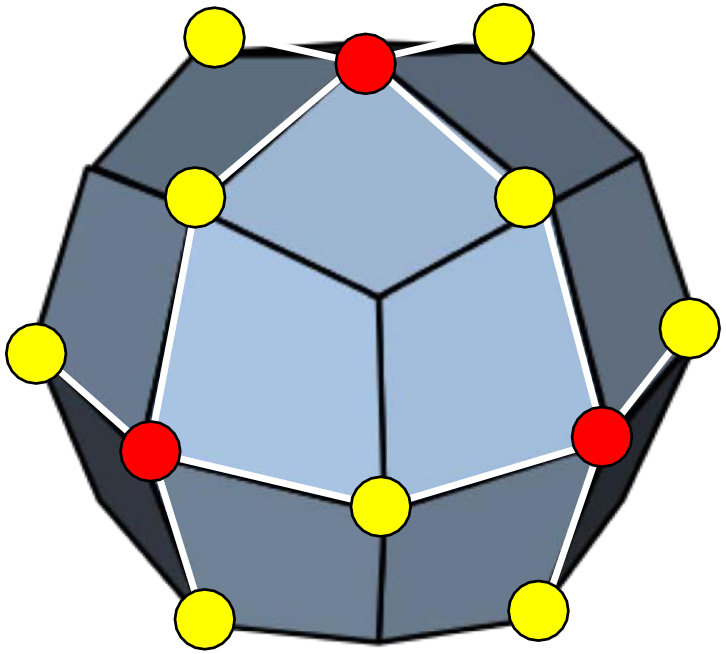
$$\frac{F + 2R + (n-3)P}{n}$$

F : Average of all n created face points adjacent to P

R : Average of all original edge midpoints touching P

Catmull-Clark Subdivision

Step 4: Move each original vertex according to new position given by:



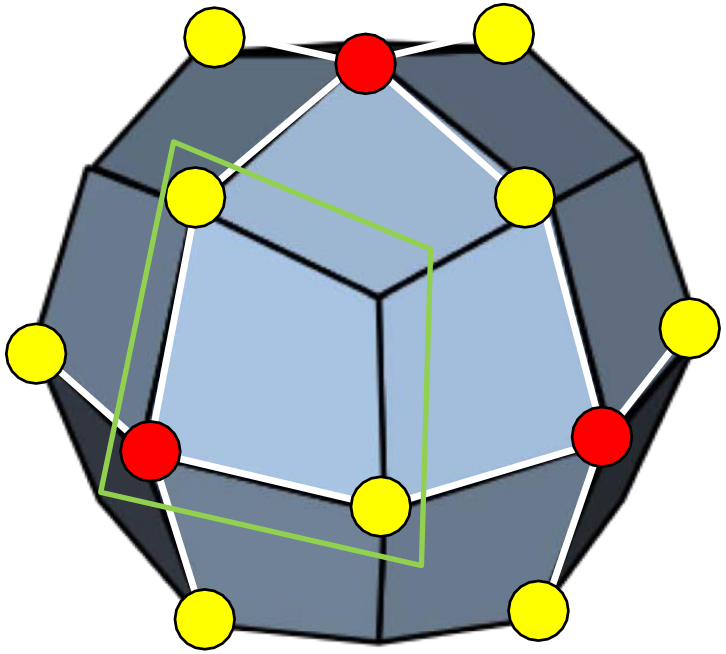
$$\frac{F + 2R + (n-3)P}{n}$$

F : Average of all n created face points adjacent to P

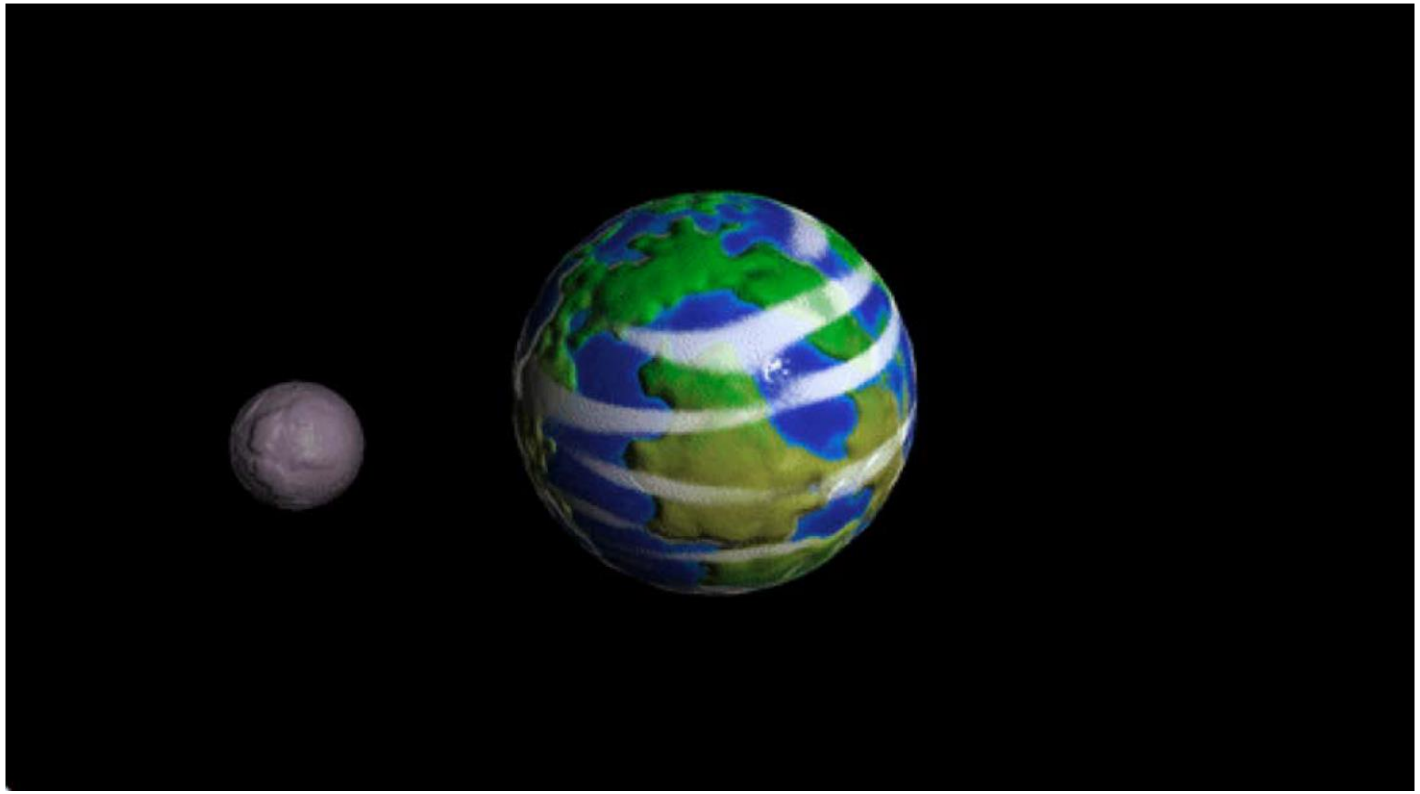
R : Average of all original edge midpoints touching P

Catmull-Clark Subdivision

Step 5: Connect up original points to make facets



Shaders



2D Affine Transformations - Translation

$$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + t_x \\ a_{21}x + a_{22}y + t_y \\ 1 \end{bmatrix}$$

$$Ax + t = b$$

Properties of 2D transforms

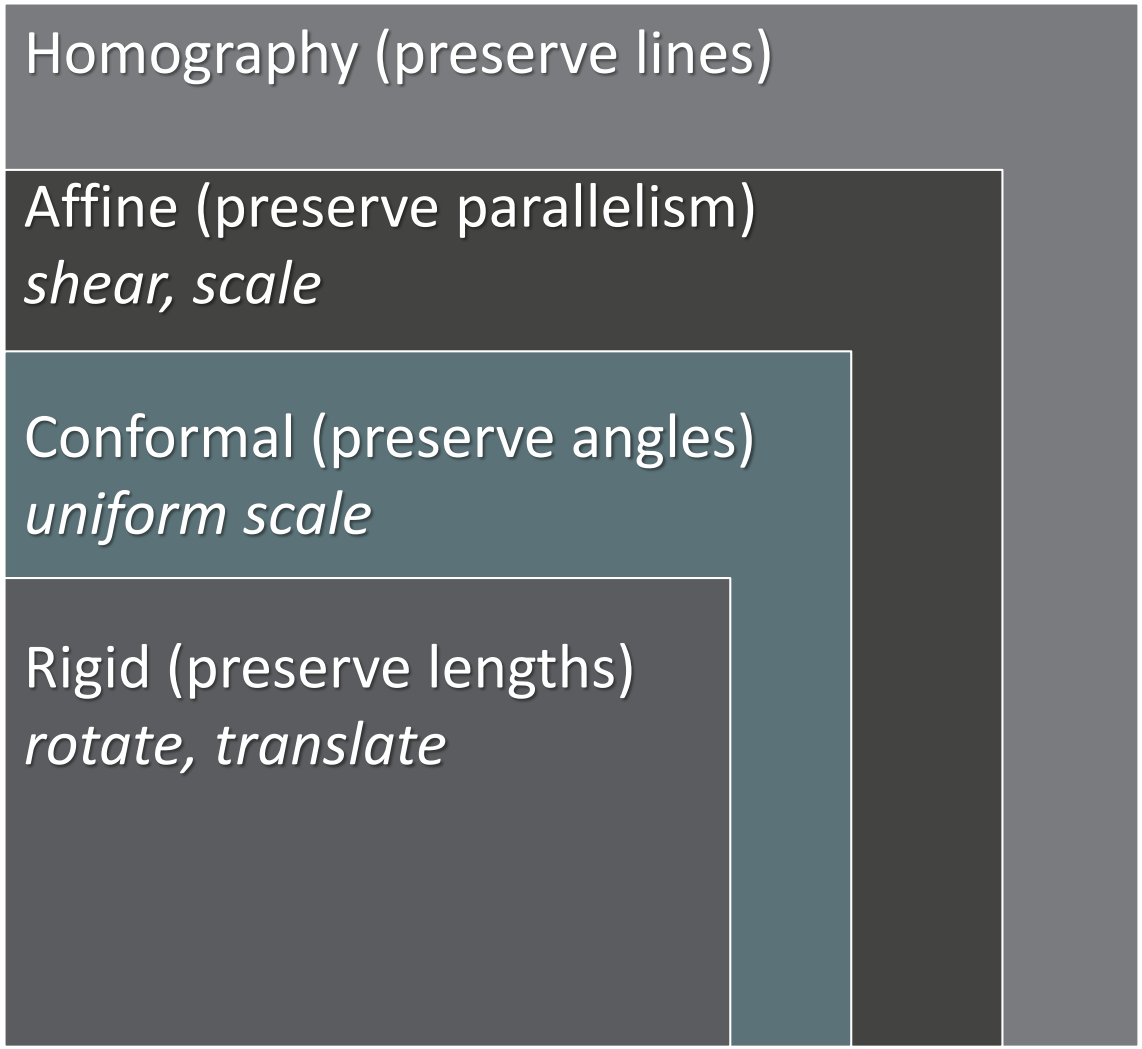
…these 3x3 transforms when invertible are called **Homographies**.
they map **lines** to **lines**.
i.e. points ***p*** on a line ***l*** transformed by a Homography ***H*** produce points ***Hp*** that also lie on a line.

…a more restricted set of transformations also preserve parallelism in lines. These are called **Affine** transforms.

…transforms that further preserve the angle between lines are called **Conformal**.

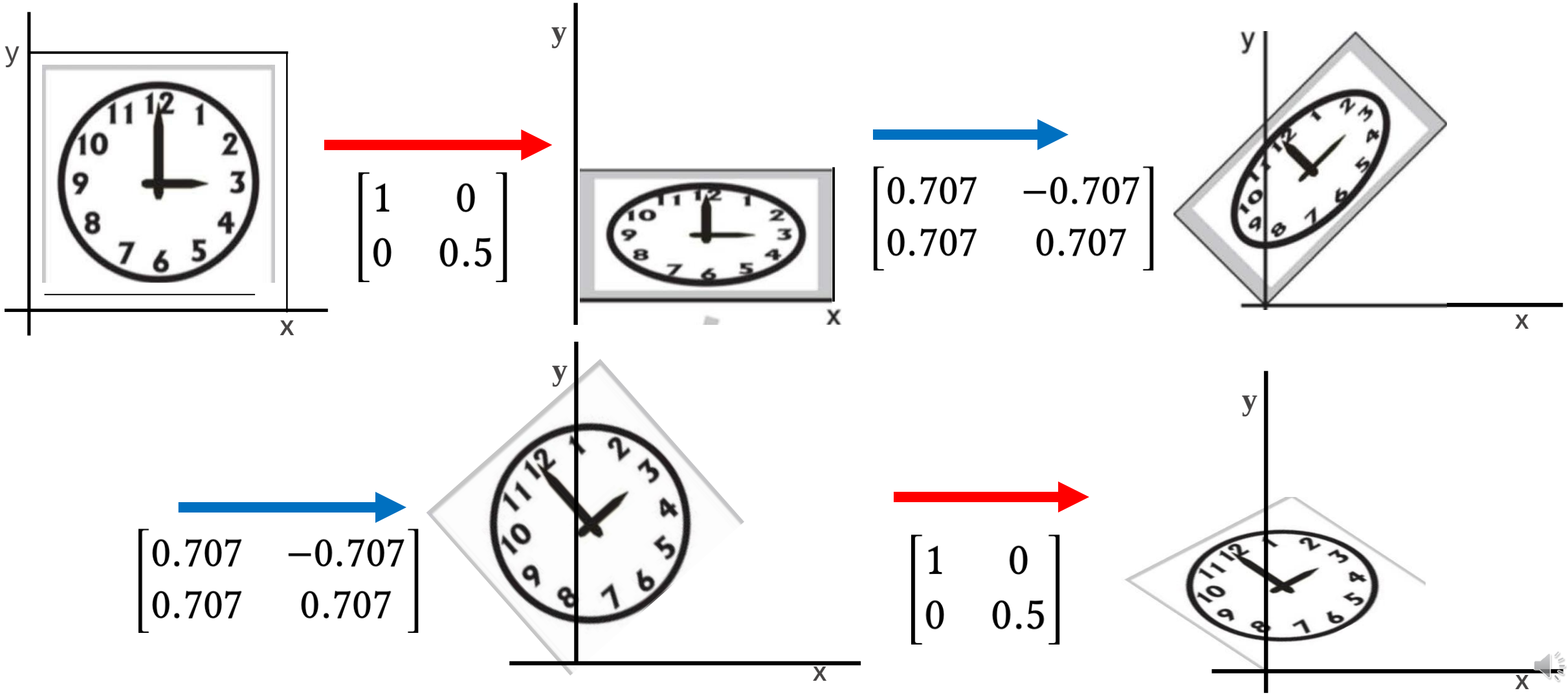
…transforms that additionally preserve the lengths of line segments are called **Rigid**.

Properties of 2D transforms

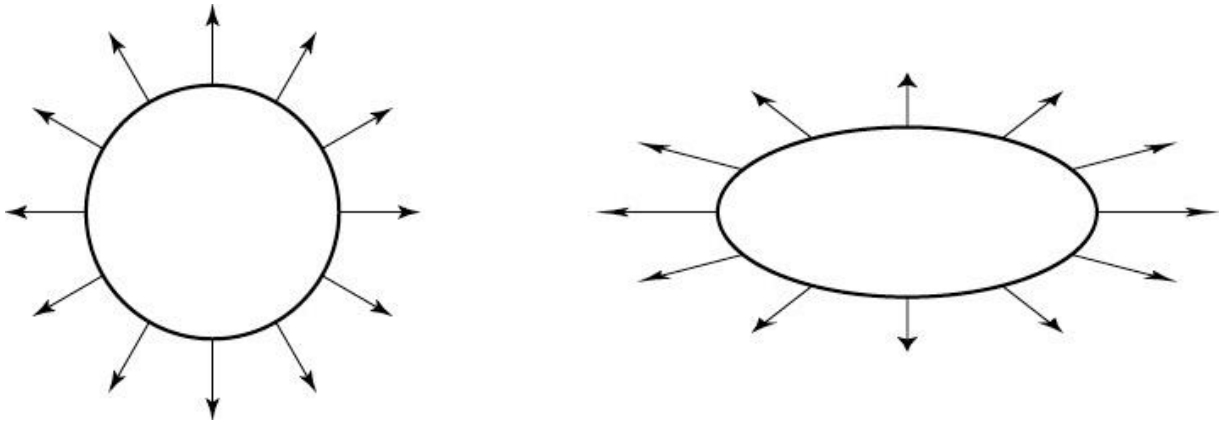


Composing Transformations

Any sequence of linear transforms can be concatenated into a single 3x3 matrix.
In general transforms DO NOT commute (some special combinations are).



Transforming tangents and normals



Tangents can be written as the difference of points on an object.
Say **t = p-q.** The transformed tangent **t'= Mp-Mq = M(p-q) = Mt.**

But **n' != Mn** for surface normals. (imagine scaling an object in x-direction)
Say **n'= Hn**. What is **H**?

$$t'^{T} \quad n' = 0$$
$$(Mt)^{T} (Hn) = 0$$
$$t^{T} (M^{T} H) n = 0$$

Remember that $t^{T} n = 0$. so… $H = (M^{T})^{-1}$

Representing 3D transforms as a 4x4 matrix

Translate a point $[x\ y\ z]^T$ by $[t_x\ t_y\ t_z]^T$:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotate a point $[x\ y\ z]^T$ by an angle t around z axis:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos t & -\sin t & 0 & 0 \\ \sin t & \cos t & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Scale a point $[x\ y\ z]^T$ by a factor $[s_x\ s_y\ s_z]^T$

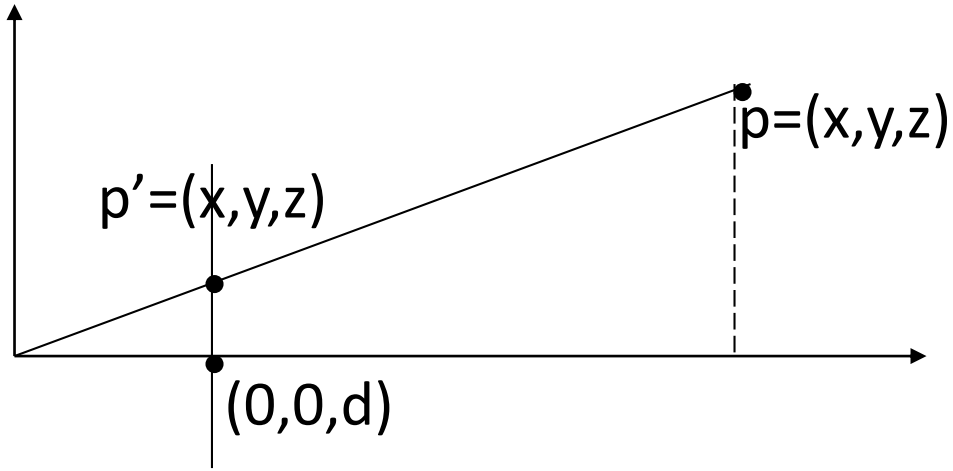
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Perspective Projection

y’= yd/z

x’= xd/z

z’=d



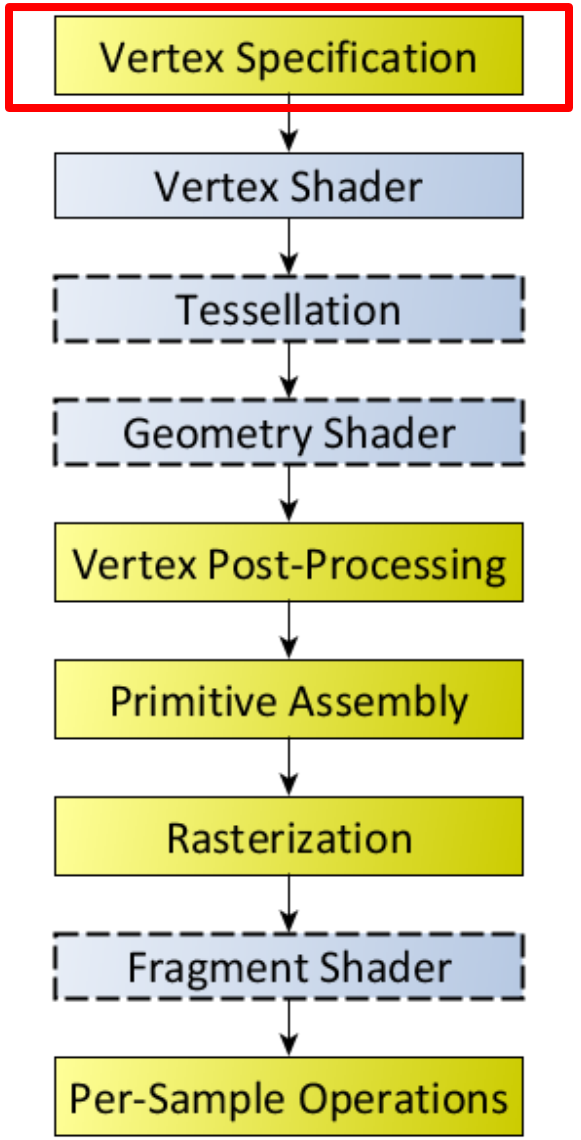
Insight: **w’=z/d**

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Modern Graphics Pipeline

A Vertex Array Object (VAO) is an object which contains one or more
Vertex Buffer Objects (VBO), designed to represent a complete rendered object.
For eg. this is a diamond consisting of four vertices as well as a color for each vertex.

```
/* We're going to create a simple diamond made from lines */
const GLfloat diamond[4][2] = {
{ 0.0, 1.0 }, /* Top point */
{ 1.0, 0.0 }, /* Right point */
{ 0.0, -1.0 }, /* Bottom point */
{ -1.0, 0.0 } }; /* Left point */

const GLfloat colors[4][3] = {
{ 1.0, 0.0, 0.0 }, /* Red */
{ 0.0, 1.0, 0.0 }, /* Green */
{ 0.0, 0.0, 1.0 }, /* Blue */
{ 1.0, 1.0, 1.0 } }; /* White */
```

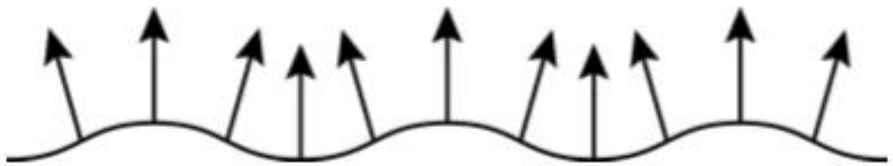


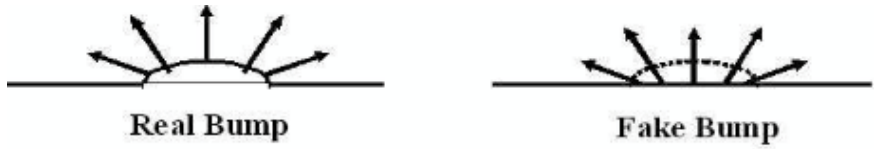
https://www.khronos.org/opengl/wiki/Tutorial2:_VAOs,_VBOs,_Vertex_and_Fragment_Shaders_(C_/_SDL)

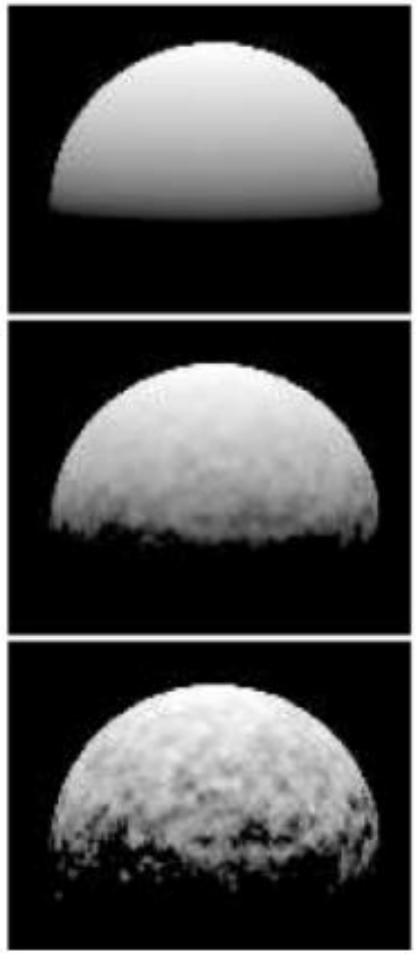
Bump | Normal Mapping

One of the reasons why we apply texture mapping:

Real surfaces are hardly flat but often rough and bumpy. These bumps cause (slightly) different reflections of the light.







Bump Mapping

Compute the perceived surface normal for the surface displaced by the bump value **B(u,v)** along its normal **n(u,v)**.

Displaced surface point $\mathbf{p_d(u,v) = p(u,v) + B(u,v)*n(u,v)}$

Displaced surface normal $\mathbf{n_d(u,v) = p'^u_d(u,v) \times p'^v_d(u,v)}$

B(u,v) is small

$$\mathbf{p'^u_d(u,v) = p'^u(u,v) + B'^u(u,v)*n(u,v) + \cancel{B(u,v)*n'^u(u,v)}}$$

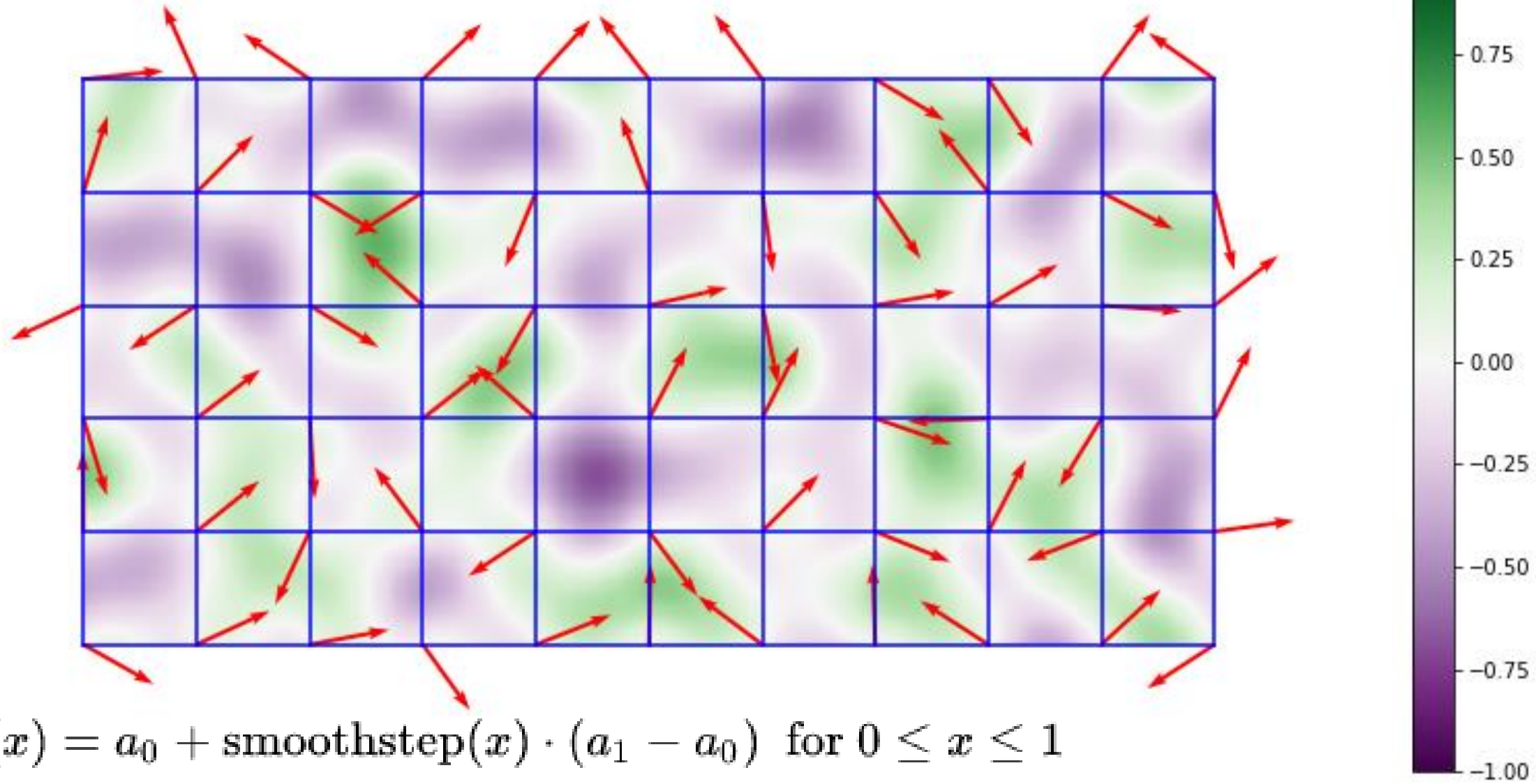
$$\mathbf{n_d = (p'^u + B'^u\, n) \times (p'^v + B'^v\, n) \quad // \text{ not showing (u,v)}}$$

$$\mathbf{n_d = p'^u \times p'^v + p'^u \times n\, B'^v + B'^u\, n \times p'^v + B'^u\, B'^v\, n \times n}$$

$$\mathbf{n_d = n + B'^v\,(p'^u \times n) + B'^u\,(n \times p'^v)}$$

B' can be precomputed by finite difference and stored as a texture.
p' can be computed analytically or by finite difference.

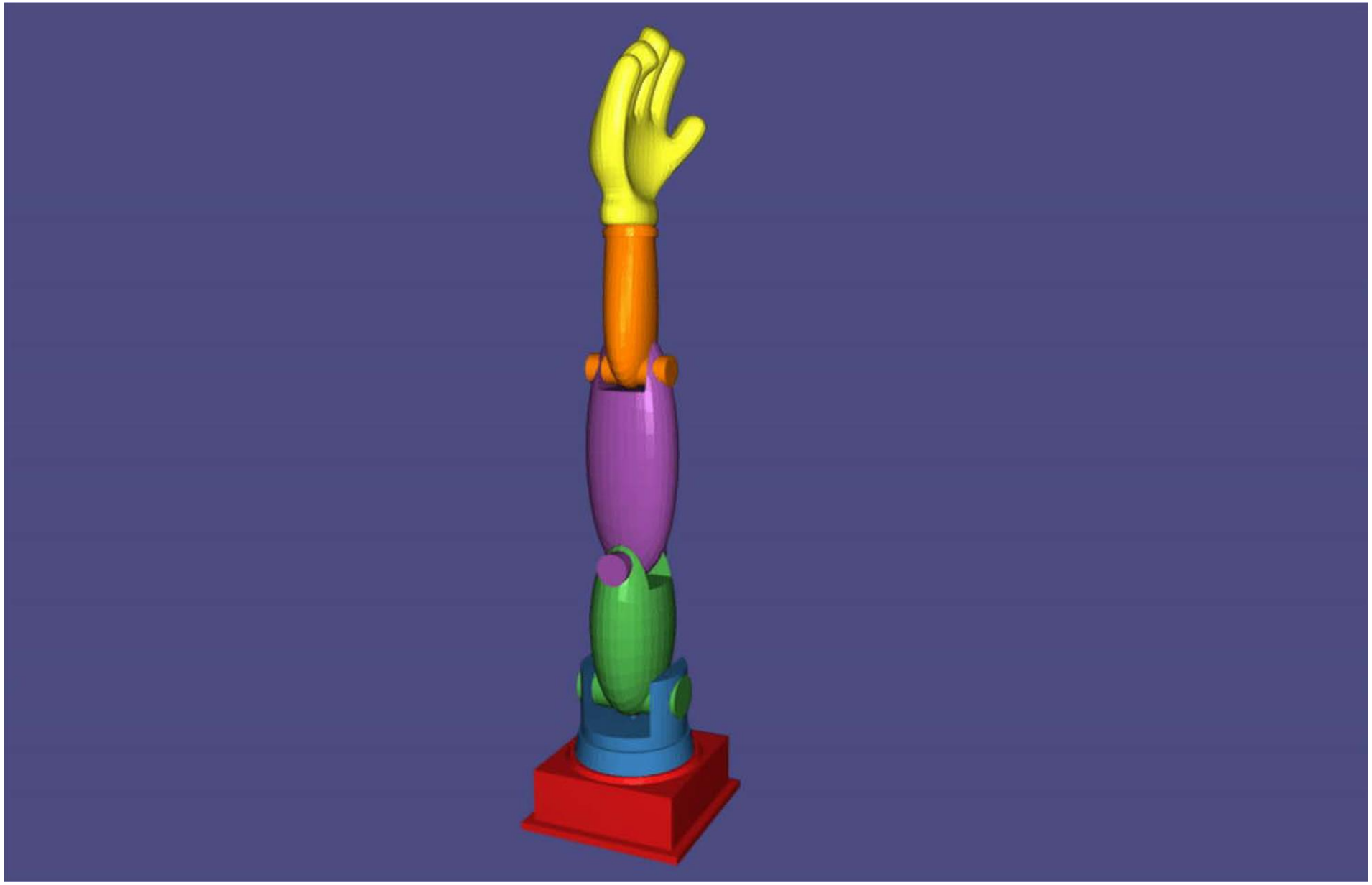
Perlin Noise



$$f(x) = a_0 + \text{smoothstep}(x) \cdot (a_1 - a_0) \quad \text{for } 0 \leq x \leq 1$$

$$\text{smoothstep}(x) = S_1(x) = \begin{cases} 0 & x \leq 0 \\ 3x^2 - 2x^3 & 0 \leq x \leq 1 \\ 1 & 1 \leq x \end{cases}$$

Kinematics



Animation Principles

Squash & Stretch
Timing
Ease-In & Ease-Out
Arcs
Anticipation
Follow-through & Secondary Motion
Overlapping Action & Asymmetry
Exaggeration
Staging
Appeal
Straight-Ahead vs. Pose-to-Pose

What can be animated?

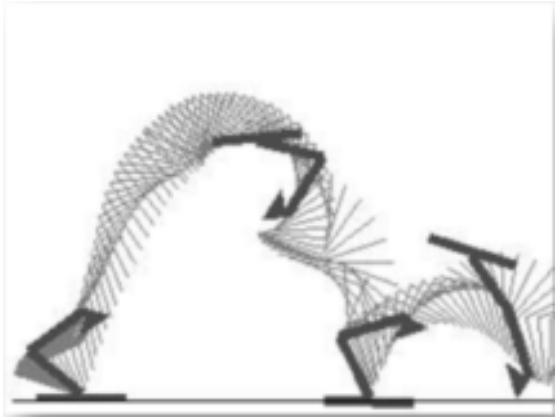
- Lights
- Camera
- **Jointed figures**
- **Skin/muscles**
- **Deformable objects**
- Clothing
- Wind/water/fire/smoke
- Hair

…any variable, Given the right time scale, almost anything…

Approaches to Animation

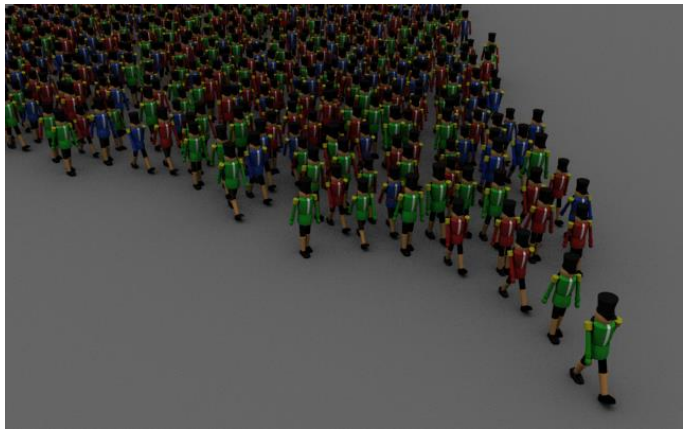
How does one make digital models move?

Keyframing



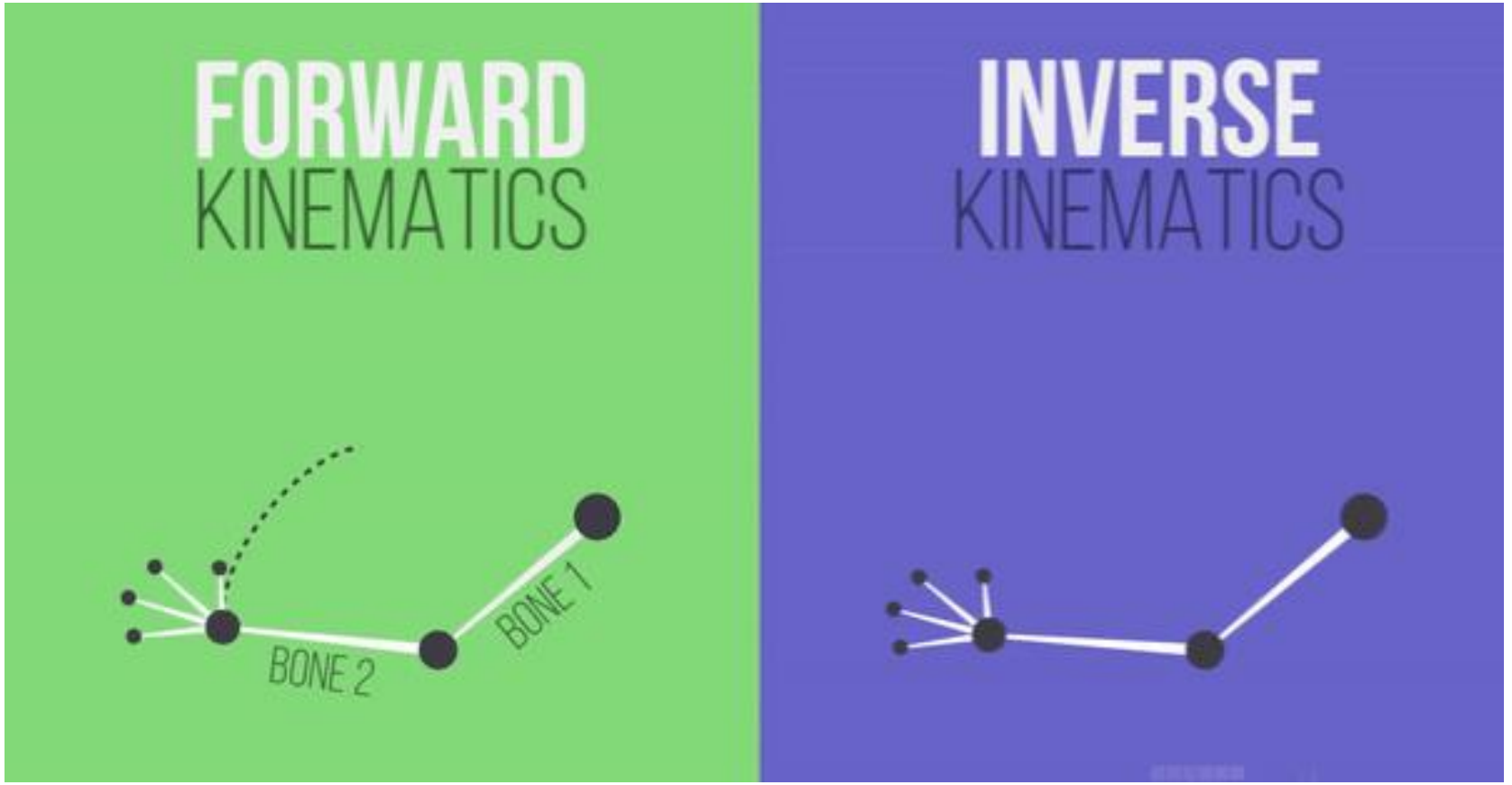
Physical simulation

Motion capture



Behavior rules

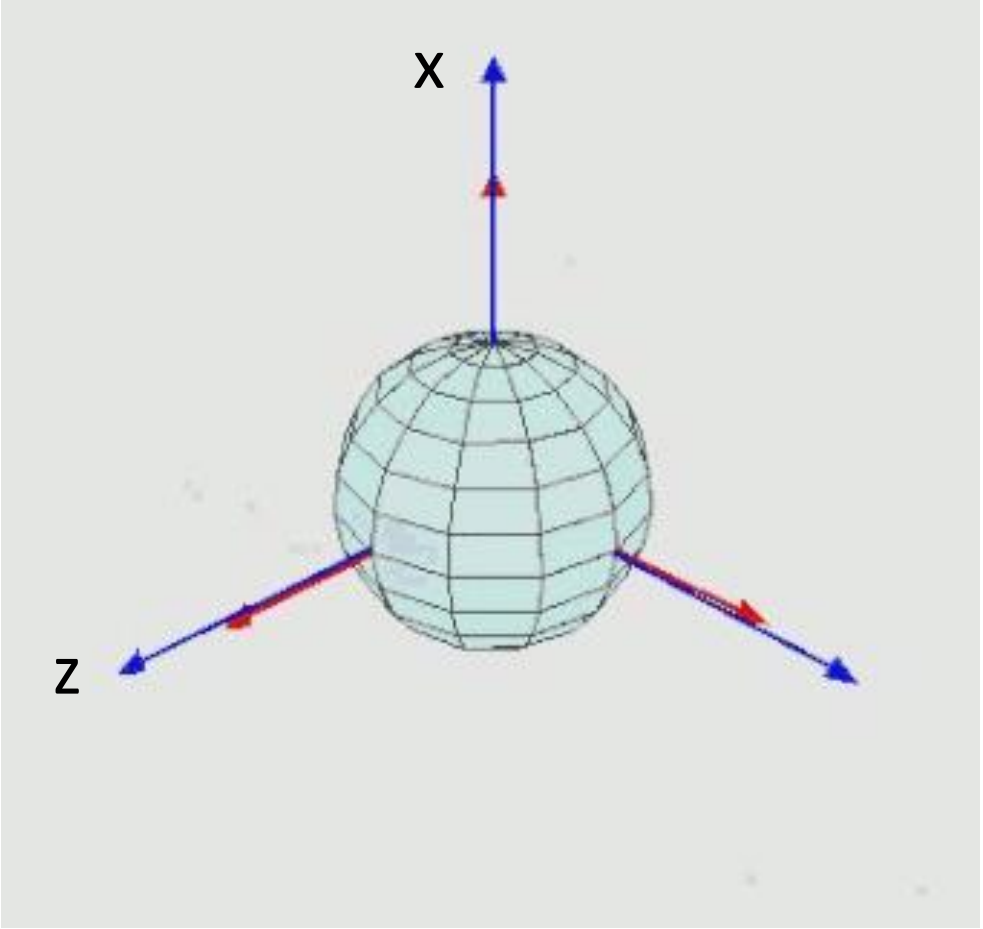
Posing a skeleton:
Forward Kinematics vs. Inverse Kinematics



https://www.youtube.com/watch?v=0a9qIj7kwiA

Euler Angle Rotations

$$R_x(\theta_3) * R_z(\theta_2) * R_x(\theta_1)$$



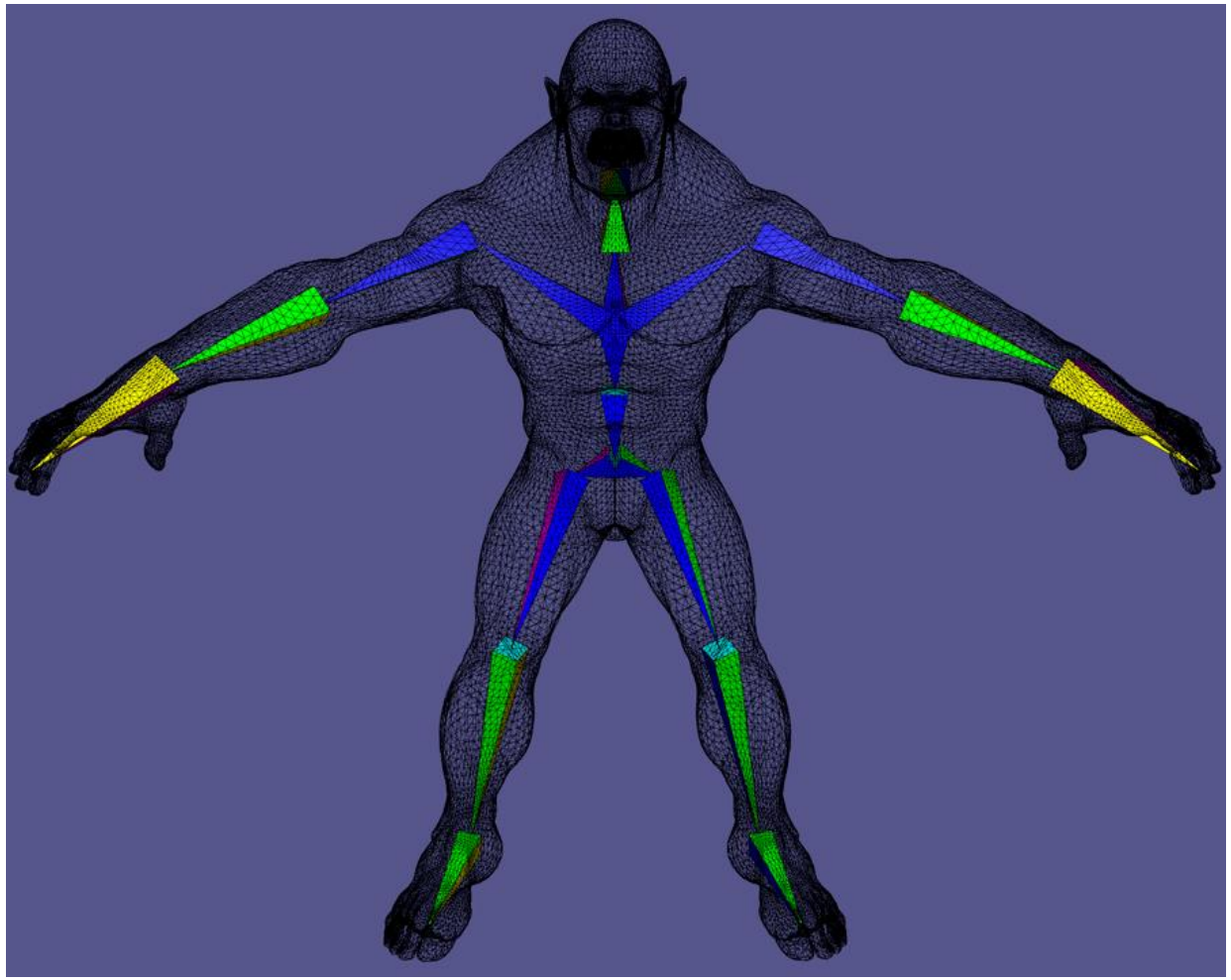
https://en.wikipedia.org/wiki/Euler_angles

https://mathworld.wolfram.com/EulerAngles.html

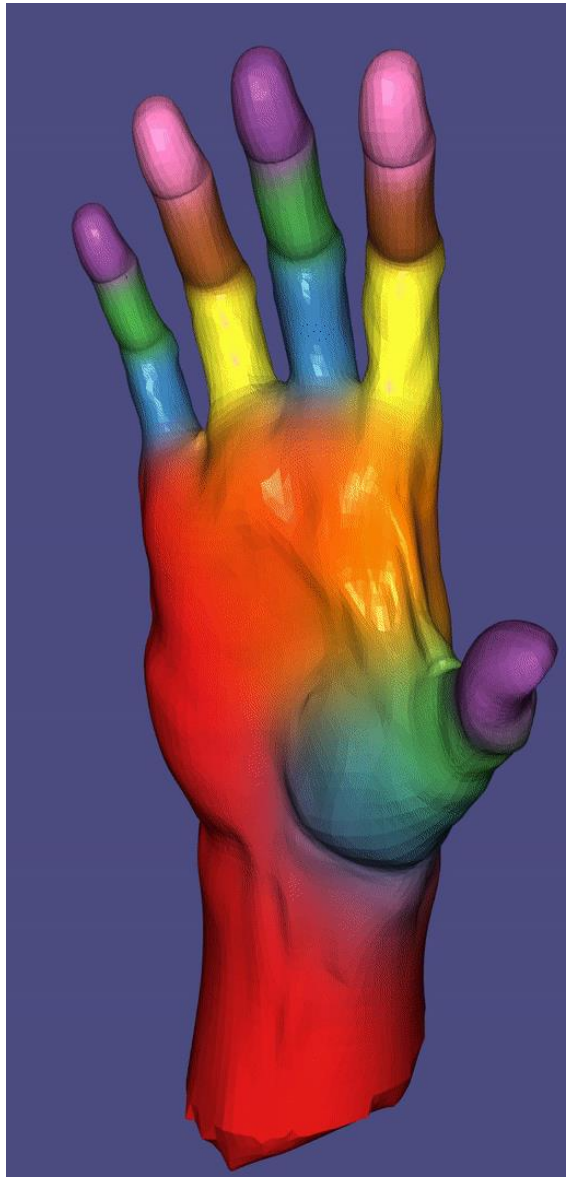
Skeletons: Forward Kinematics

$$\mathbf{T}_i = \mathbf{T}_{p_i} \begin{pmatrix} \widehat{\mathbf{T}}_i \\ 0\,0\,0\,1 \end{pmatrix} \begin{pmatrix} \overline{\mathbf{R}}_i & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0\,0\,0 \end{matrix} & 1 \end{pmatrix} \begin{pmatrix} \widehat{\mathbf{T}}_i \\ 0\,0\,0\,1 \end{pmatrix}^{-1}$$

$$\mathbf{T}_i = \mathbf{T}_{p_i} \widehat{\mathbf{T}}_i \begin{pmatrix} \mathbf{R}_x(\theta_{i3}) & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0\,0\,0 \end{matrix} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R}_z(\theta_{i2}) & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0\,0\,0 \end{matrix} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R}_x(\theta_{i1}) & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0\,0\,0 \end{matrix} & 1 \end{pmatrix} \widehat{\mathbf{T}}_i^{-1}$$



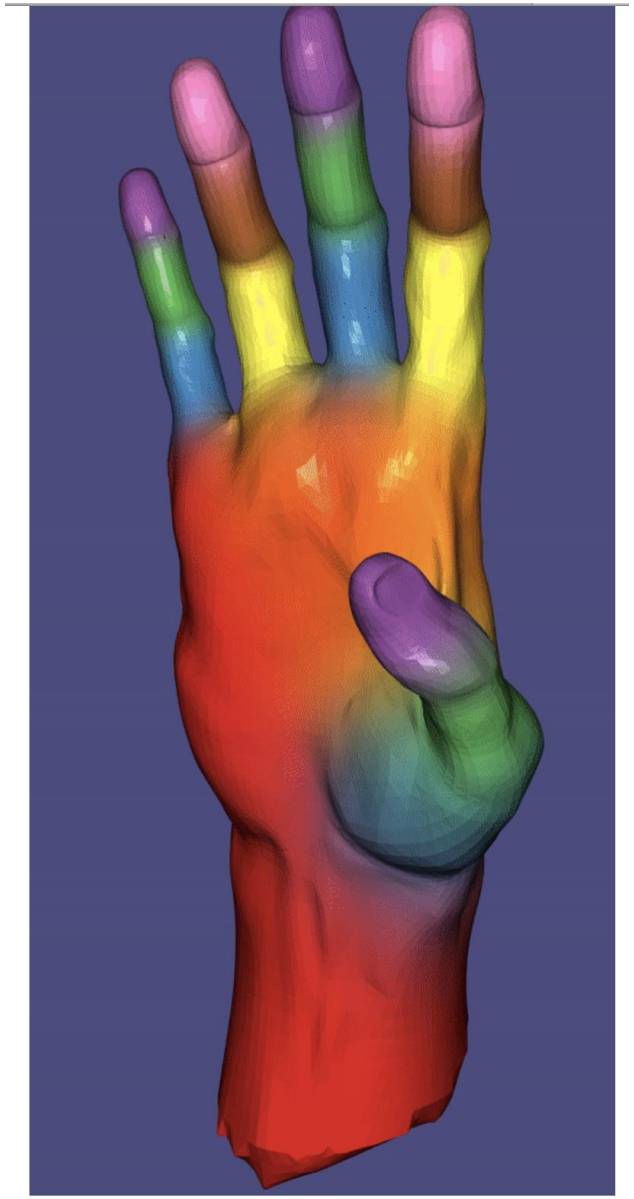
Linear Blend Skinning



$$\mathbf{v}_j = \sum_{i=1}^{\#\text{bones}} w_{ij} \mathbf{T}_i \hat{\mathbf{v}}_j$$

Specifying Keyframes

Time = 0



Time = 10s

Poses are generated by specifying rotations of bones

Each pose can be represented as

$$\theta = \begin{pmatrix} \theta_{11} \\ \theta_{11} \\ \theta_{11} \\ \vdots \\ \theta_{n1} \\ \theta_{n2} \\ \theta_{n3} \end{pmatrix}$$

Catmull-Romm Spline

After solving and rearranging we end up with

$$\mathbf{c}(t) = (2t^3 - 3t^2 + 1)\theta_0 + (t^3 - 2t^2 + t)\mathbf{m}_0 + (-2t^3 + 3t^2)\theta_1 + (t^3 - t^2)\mathbf{m}_1$$

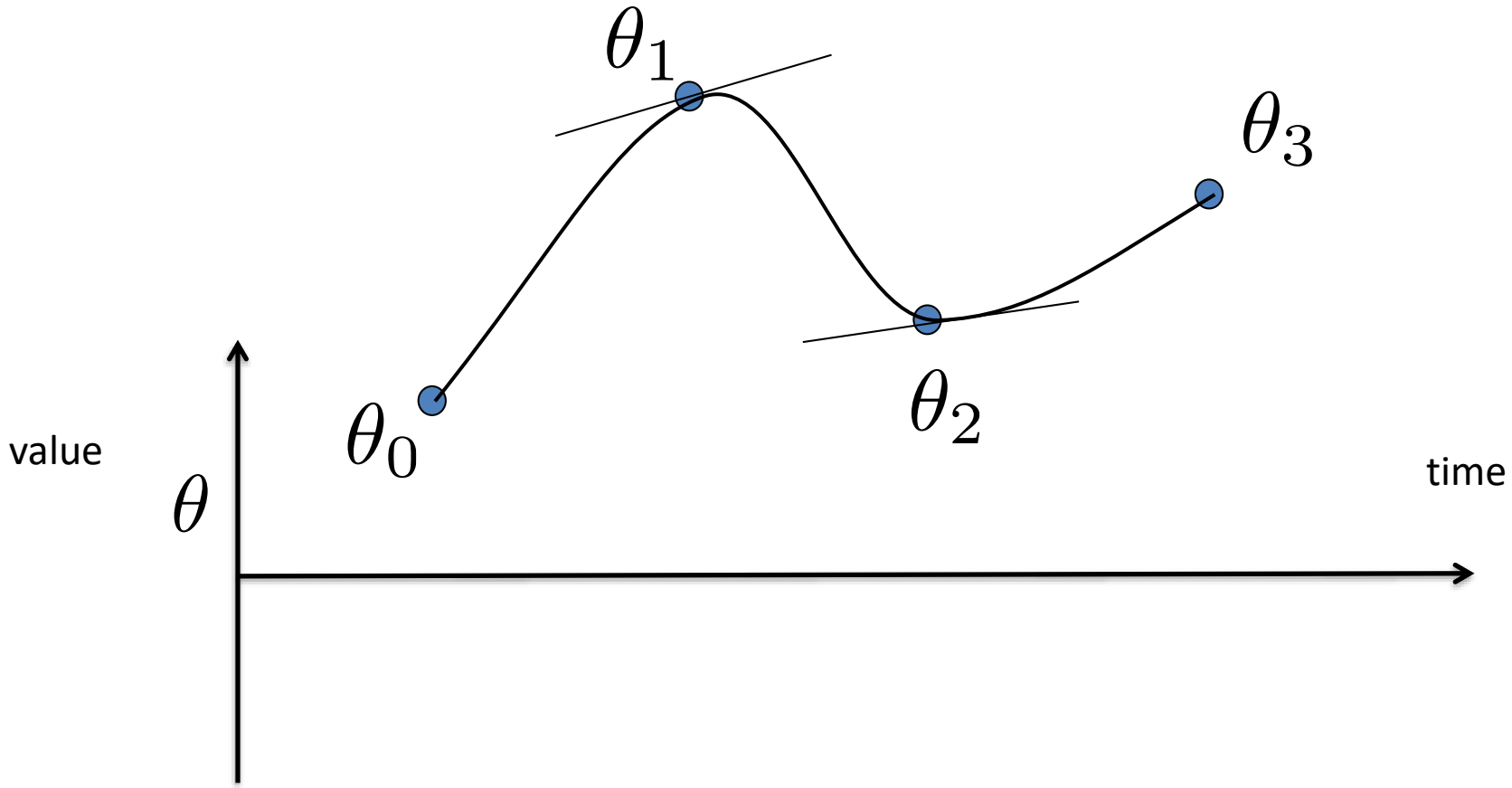
Remember this is for t = 0 to t = 1.

For arbitrary intervals substitute

$$t = (t' - t_0)/(t_1 - t_0)$$

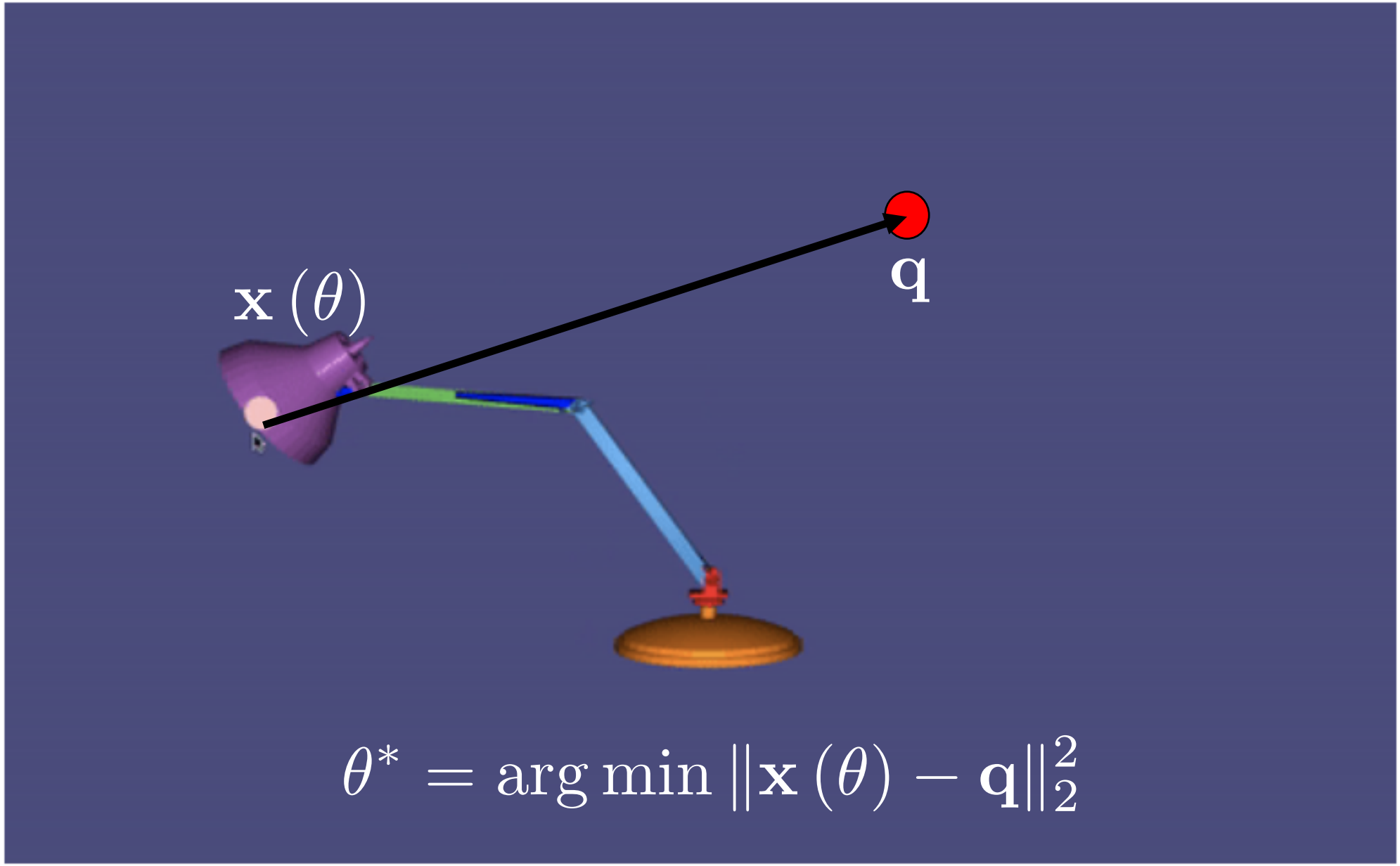
Catmull-Romm Spline

Catmull-Romm chooses the tangents using “Finite Differences”



$$\mathbf{m}_k = \frac{\theta_{k+1} - \theta_k}{t_{k+1} - t_k}$$

Inverse Kinematics



Skeletons: Inverse Kinematics

Closeness energy can be measured the squared distance between the pose tip x_b of some bone b and a desired goal location $q \in R^3$.

$$E(\mathbf{x}_b) = \|\mathbf{x}_b - \mathbf{q}\|^2.$$

Given pose vector a, the bone tip x_b is:

$$\mathbf{x}_b(\mathbf{a}) = \mathbf{T}_b \widehat{\mathbf{d}}_b$$

Now given any number of end-effectors $b_1,..b_k$:

$$\min_{\mathbf{a}} \underbrace{\sum_{i=1}^{k} \|\mathbf{x}_{b_i}(\mathbf{a}) - \widehat{\mathbf{x}}_{b_i}\|^2}_{E(\mathbf{x}_b(\mathbf{a}))}$$

And we impose some joint angle limits: $\min_{\mathbf{a}^{\min} \le \mathbf{a} \le \mathbf{a}^{\max}} E(\mathbf{x}_b(\mathbf{a}))$

Minimizing this energy is a non-linear least squares problem.

Skeletons: Inverse Kinematics Minimization

Projected Gradient Descent:
Start with an initial pose *a*, and move in direction of decrease in *E*,
project the pose to stay within limits and iterate towards solution.

$$\mathbf{a} \leftarrow \mathbf{a} - \sigma \left(\frac{dE(\mathbf{x}(\mathbf{a}))}{d\mathbf{a}} \right)^T$$

$$\mathbf{a} \leftarrow \mathbf{a} - \sigma \left(\frac{d\mathbf{x}(\mathbf{a})}{d\mathbf{a}} \right)^T \left(\frac{dE(\mathbf{x})}{d\mathbf{x}} \right)$$ chain rule

$$\frac{dE}{d\mathbf{a}} \in \mathbb{R}^{|\mathbf{a}|}, \frac{dE}{d\mathbf{x}} \in \mathbb{R}^{|\mathbf{x}|}, \text{ and } \frac{d\mathbf{x}}{d\mathbf{a}} \in \mathbb{R}^{|\mathbf{x}| \times |\mathbf{a}|}$$

$$\mathbf{J} = \frac{d\mathbf{x}}{d\mathbf{a}}.$$ also known as Jacobian measures the change in *x* for changes in joint angles *a*,

J can be computed using Finite Differences: $$\mathbf{J}_{i,j} \approx \frac{\mathbf{x}_i(\mathbf{a} + h\delta_j)}{h}. \qquad h = 10^{-7}$$

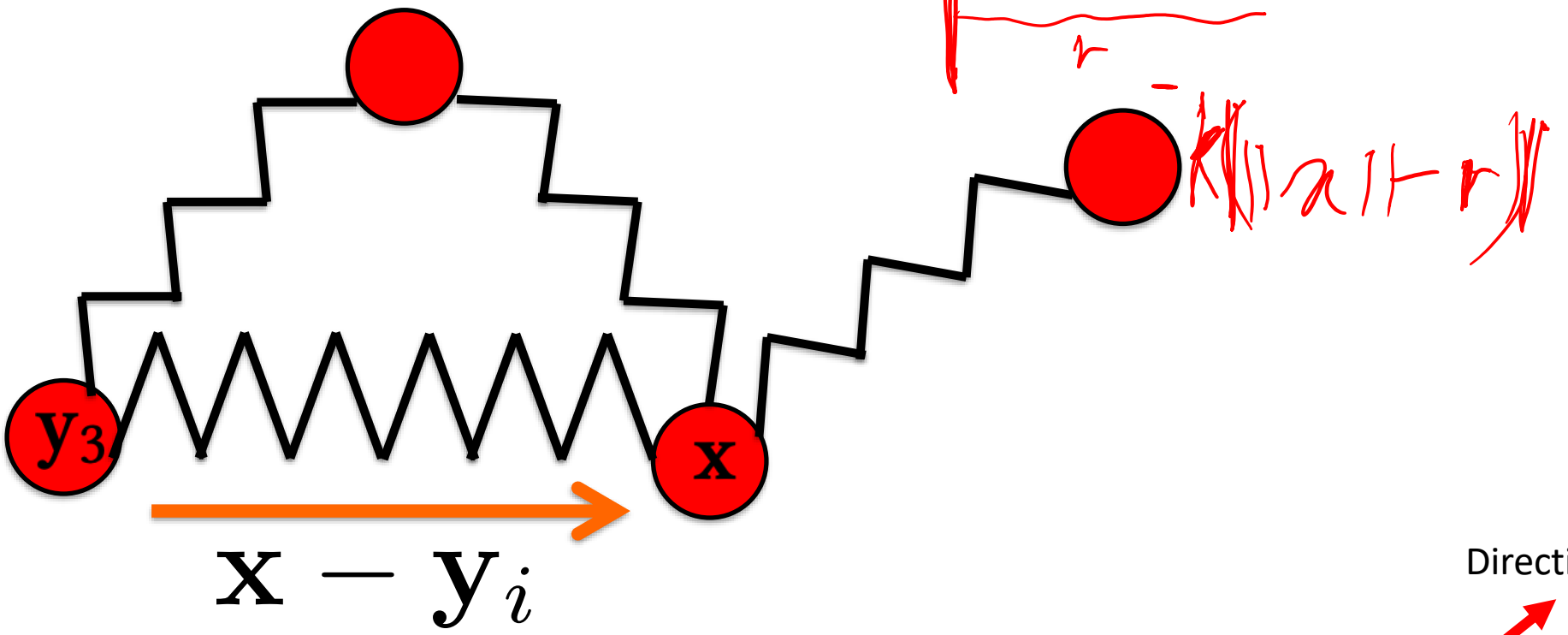
$\left(\frac{dE(\mathbf{x})}{d\mathbf{x}} \right)$ is gradient of $$\sum_{i=1}^{k} \|\mathbf{x}_{b_i}(\mathbf{a}) - \widehat{\mathbf{x}}_{b_i}\|^2$$

Project to within limits: $$\mathbf{a}_i \leftarrow \max[\mathbf{a}_i^{\min}, \min[\mathbf{a}_i^{\max}, \mathbf{a}_i]].$$

Find a good step that lowers energy: $$E(\text{proj}(\mathbf{a} + \sigma \, \Delta \, \mathbf{a})) < E(\mathbf{a}).$$

Mass-Springs

The Mass-Spring System



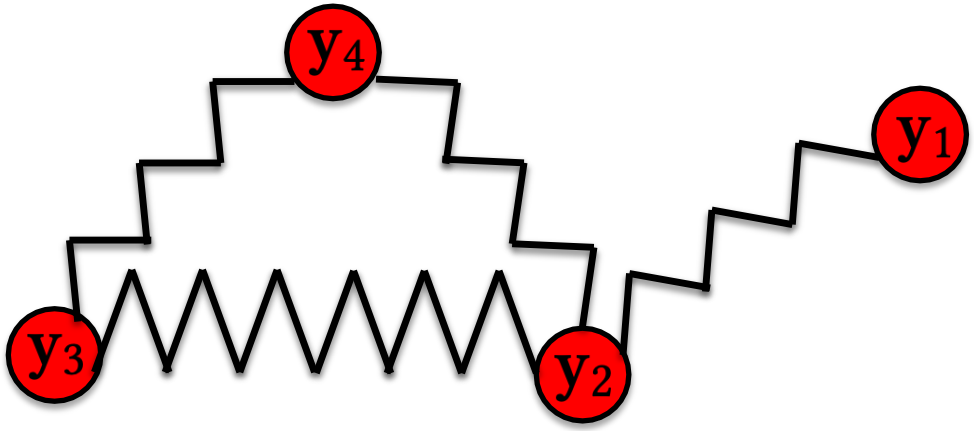
Direction of Spring

Rest Length

$$\mathbf{f}_x = -k(\|\mathbf{x} - \mathbf{y_i}\| - r_{ix}) \frac{\mathbf{x} - \mathbf{y_i}}{\|\mathbf{x} - \mathbf{y_i}\|}$$

Change in Length

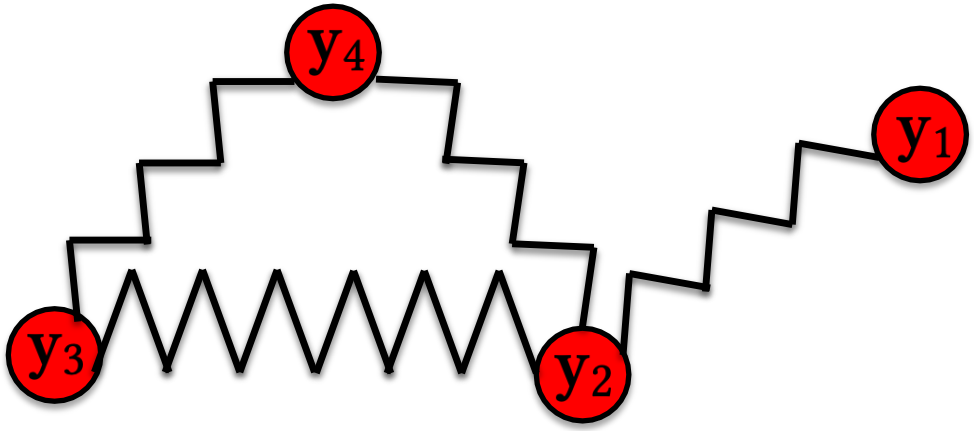
Newton's Second Law: System of Equations



$$\begin{pmatrix} m_1 \cdot I & 0 & 0 & 0 \\ 0 & m_2 \cdot I & 0 & 0 \\ 0 & 0 & m_3 \cdot I & 0 \\ 0 & 0 & 0 & m_4 \cdot I \end{pmatrix} \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \\ \mathbf{a}_4 \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \\ \mathbf{f}_4 \end{pmatrix}$$

Mass Matrix $\qquad \mathbf{a}(t) \qquad \mathbf{f}(t)$

Time Integration



Need to Discretize!

$$M\frac{d^2\mathbf{y}(t)}{dt^2} = \mathbf{f}(\mathbf{y}(t))$$

Use Finite Differences!

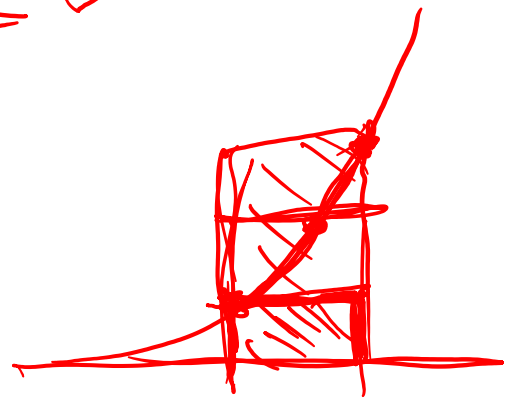
$$\frac{d^2\mathbf{y}(t)}{dt^2} \approx \frac{\mathbf{y}^{t+1} - 2\mathbf{y}^t + \mathbf{y}^{t-1}}{\Delta t^2}$$

Time Integration: Explicit vs. Implicit

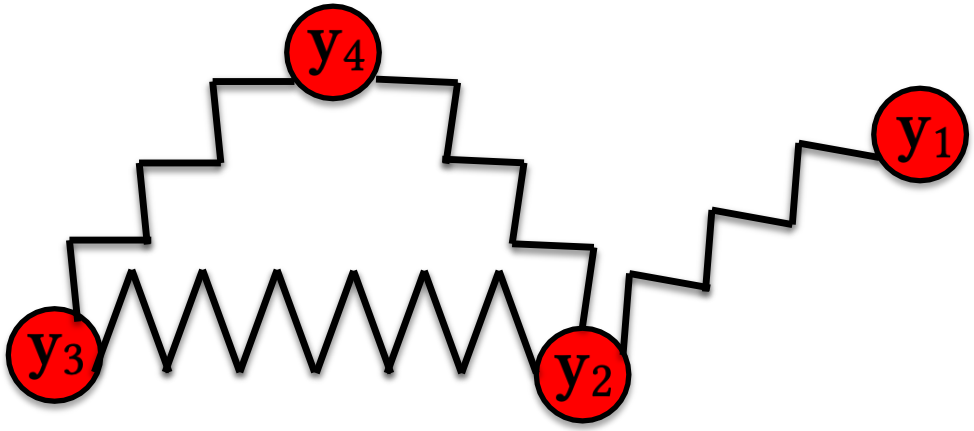
$$M\frac{d^2\mathbf{y}(t)}{dt^2} = \mathbf{f}(\mathbf{y}(t))$$

Explicit: $y_{t+dt} = g(y_t)$. Future state y_{t+dt} is an explicit equation of current state y_t and dt.

Implicit: $h(y_t, y_{t+dt})=0$. Future state y_{t+dt} is an implicit equation.



Implicit Time Integration



$$M\left(\frac{\mathbf{y}^{t+1} - 2\mathbf{y}^{t} + \mathbf{y}^{t-1}}{\Delta t^2}\right) = \mathbf{f}(\mathbf{y}^{t+1})$$

$$M\mathbf{y}^{t+1} = M\left(2\mathbf{y}^{t} - \mathbf{y}^{t-1}\right) + \Delta t^2 \mathbf{f}\left(\mathbf{y}^{t+1}\right)$$

Goal: Solve for $\mathbf{y}^{t+1}$

Energy

$$\mathbf{y}^{t+1} = \underset{\mathbf{y}}{\operatorname{argmin}} \underbrace{\left(\sum_{ij} \frac{1}{2} k(\|\mathbf{y}_i - \mathbf{y}_j\| - r_{ij})^2\right) - \frac{\Delta t^2}{2}\left(\sum_i m_i \left(\frac{\mathbf{y}_i - 2\mathbf{y}_i^t + \mathbf{y}_i^{t-1}}{\Delta t^2}\right)^2\right) - \left(\sum_i \mathbf{y}_i^\top \mathbf{f}_i^{\text{ext}}\right)}_{E(\mathbf{y})}$$

External forces

Potential energy force

$$V(\mathbf{y}_i, \mathbf{x}) = \frac{1}{2} k(\|\mathbf{y}_i - \mathbf{x}\| - r_{ix})^2$$

$0.5*ma^2$

Kinetic energy-like

$$\mathbf{a}_i^t = \ddot{\mathbf{y}}_i^t = \frac{d^2\mathbf{y}_i(t)}{dt^2} \approx \frac{\mathbf{y}_i^{t+1} - 2\mathbf{y}_i^t + \mathbf{y}_i^{t-1}}{\Delta t^2}$$

Observation!

$$(\|\mathbf{y}_i - \mathbf{y}_j\| - r_{ij})^2 = \min_{\mathbf{d}_{ij} \in \mathbb{R}^3, \|\mathbf{d}\| = r_{ij}} \|(\mathbf{y}_i - \mathbf{y}_j) - \mathbf{d}_{ij}\|^2$$

$$\mathbf{y}^{t+1} = \underset{\mathbf{y}}{\operatorname{argmin}} \underbrace{\left(\sum_{ij} \frac{1}{2} k \|(\mathbf{y}_i - \mathbf{y}_j) - \mathbf{d}_{ij}\|^2 \right)}_{E_2(\mathbf{y})} - \frac{\Delta t^2}{2} \underbrace{\left(\sum_i m_i \left(\frac{\mathbf{y}_i - 2\mathbf{y}_i^t + \mathbf{y}_i^{t-1}}{\Delta t^2} \right)^2 \right) - \left(\sum_i \mathbf{y}_i^\top \mathbf{f}_i^{\text{ext}} \right)}_{E_1(\mathbf{y})}}_{\tilde{E}(\mathbf{y})}$$

Quadratic!

Local-Global Solvers for Mass-Spring Systems

WHILE Not done

    For Each Spring

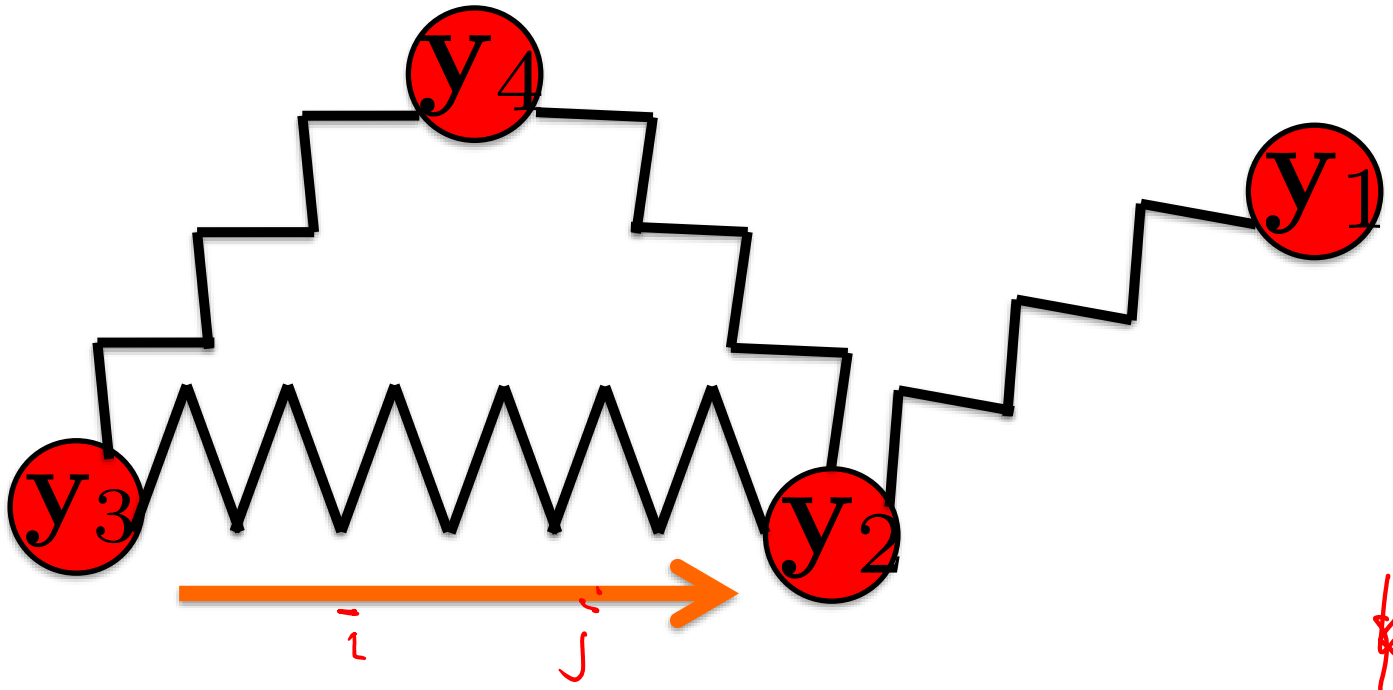
        Local Optimization

    Global Optimization

END

Now we can start defining these steps for mass-springs

The Global Step



$$\mathbf{y} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \\ \mathbf{y}_4 \end{pmatrix}$$

$k^{th} = y_i - y_j$

$$\Delta\mathbf{y} = \underbrace{\begin{pmatrix} I & -I & 0 & 0 \\ 0 & I & -I & 0 \\ 0 & I & 0 & -I \\ 0 & -I & I & -I \end{pmatrix}}_{G} \underbrace{\begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \\ \mathbf{y}_4 \end{pmatrix}}_{m \times n}$$

Each row is a spring

Local-Global Solvers for Mass-Spring Systems

WHILE Not done

//Local Steps

For Each Spring

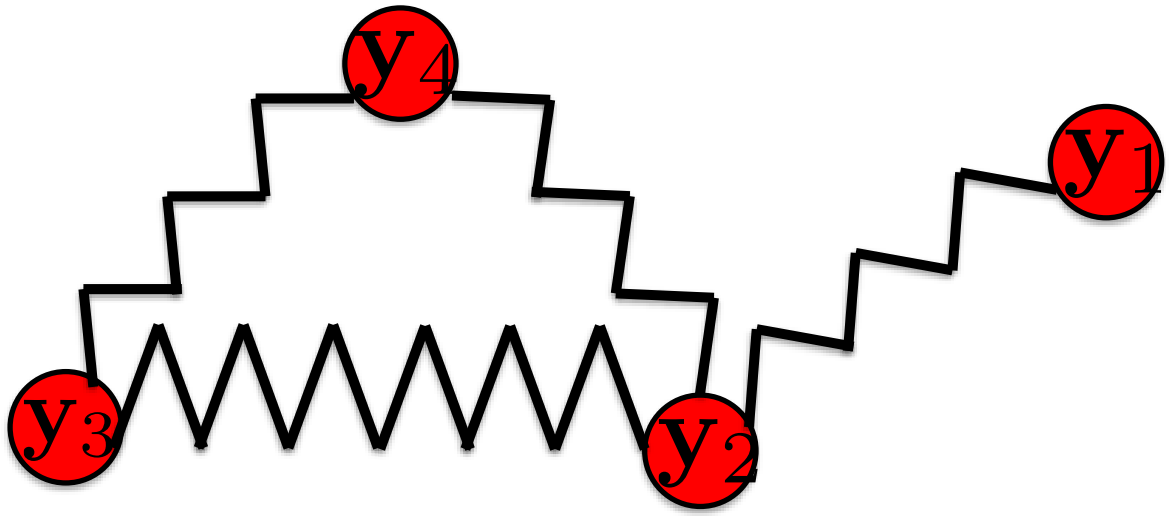
$$E_{ij} = \arg\min_{\mathbf{d}_{ij}, |\mathbf{d}_{ij}| = r_{ij}} \frac{k}{2} \|\mathbf{y}_i - \mathbf{y_j} - \mathbf{d}_{ij}\|^2$$

//Global Step

Solve $(M + \Delta t^2 k G^T G)\mathbf{y} = (M\mathbf{b} + \Delta t^2 k \mathbf{G}^T \mathbf{d})$

END

Fixed Points via Projection



$$(M + \Delta t^2 k G^T G)\mathbf{y} = (M\mathbf{b} - \Delta t^2 k \mathbf{G}^T \mathbf{d})$$

Substituting $\mathbf{y} = P\tilde{\mathbf{y}} + \mathbf{c}$ in $A\mathbf{y} = \mathbf{f}$

Too many rows now … $AP\tilde{\mathbf{y}} = \mathbf{f} - A\mathbf{c}$

$$P^T A P \tilde{\mathbf{y}} = P^T(\mathbf{f} - A\mathbf{c})$$

…Rebuild $\mathbf{y}$

This is the processing behind the pixels,
the math behind the magic!