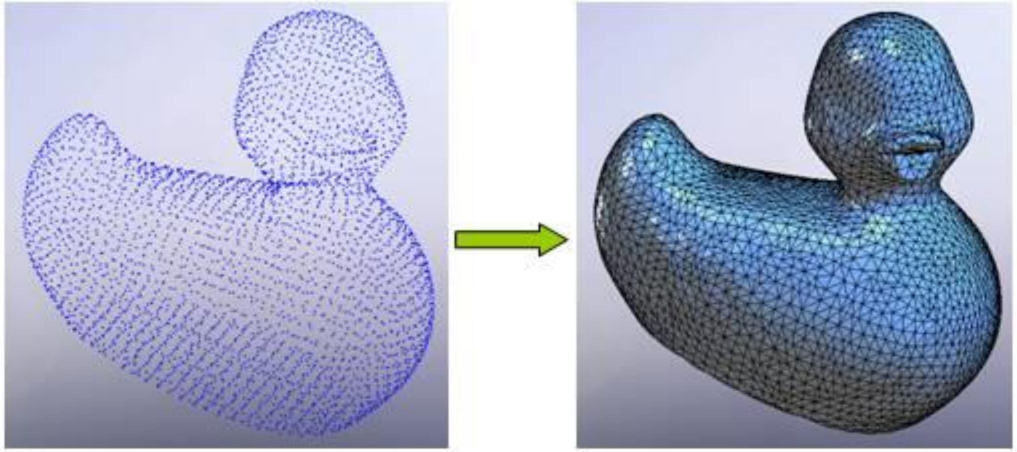
# Surface Reconstruction Using the Ball-Pivoting Algorithm

Allen Zeng

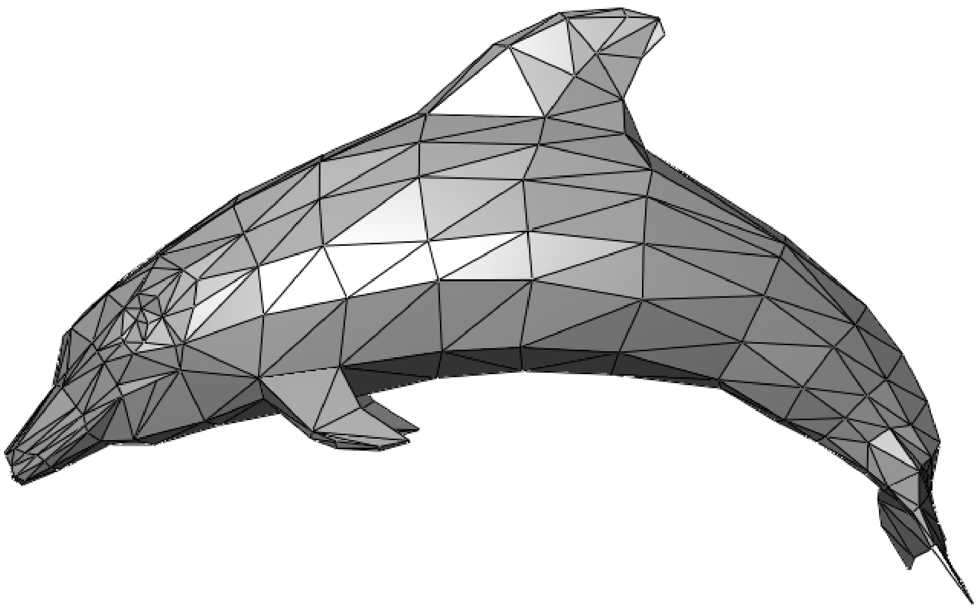
# Surface Mesh Reconstruction

- Goal: Given a real world object, we want to create a digital model of it.
- Scanners can sample points from physical objects
- Each point has an associated surface normal, defining which direction is “up” relative to a surface
- Polygon meshes are useful for conducting computer simulations of different scenarios

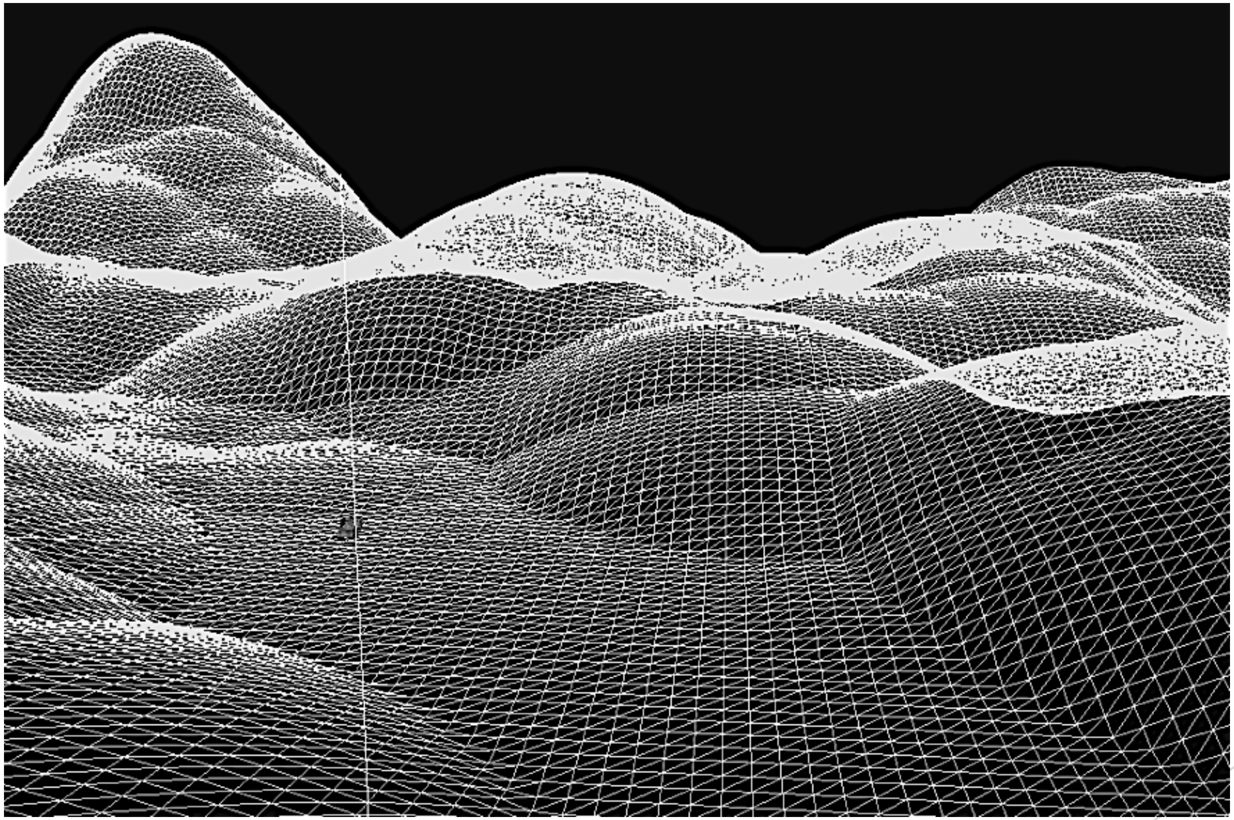


# What is a Polygon Mesh?

- A collection of vertices, edges and faces that defines the shape of a polyhedral object
- The faces are usually triangles, quadrilaterals, or other simple polygons
- Useful for simulations: ray-tracing, thermodynamics, collision detection, physical dynamics

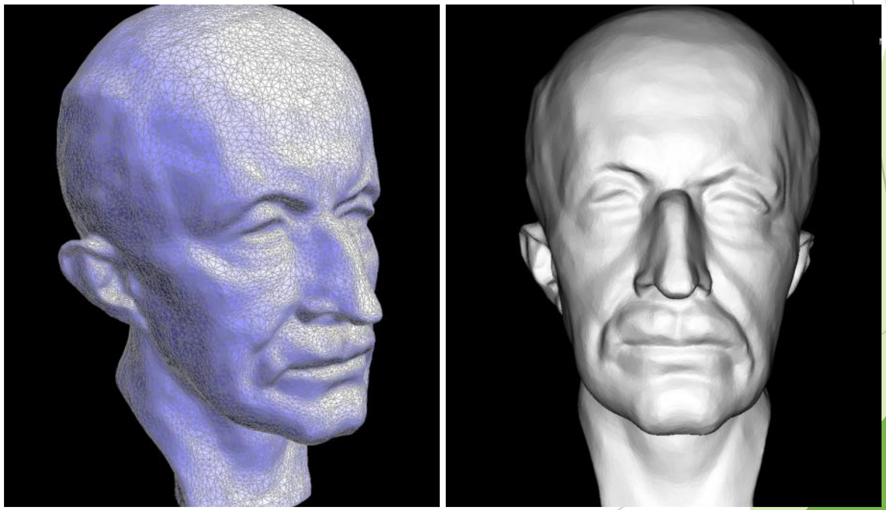


# Applications of Surface Reconstruction

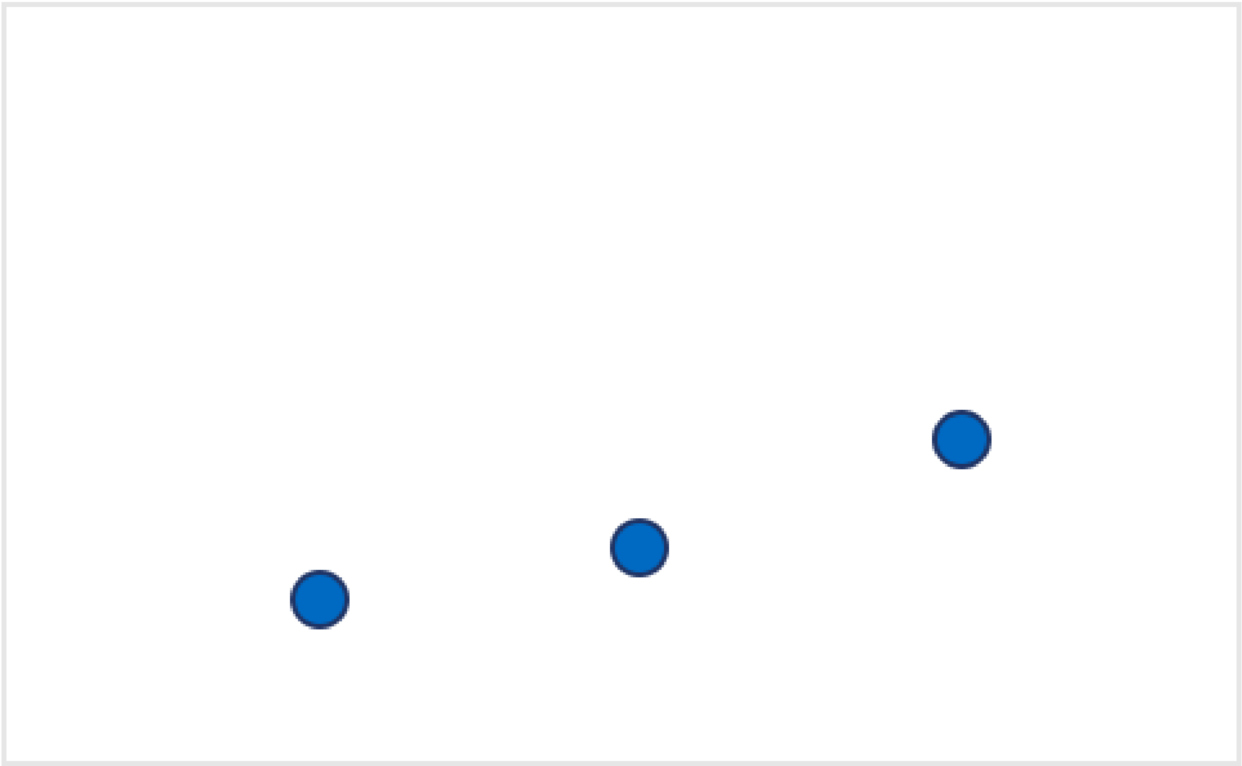


# Applications of Surface Reconstruction

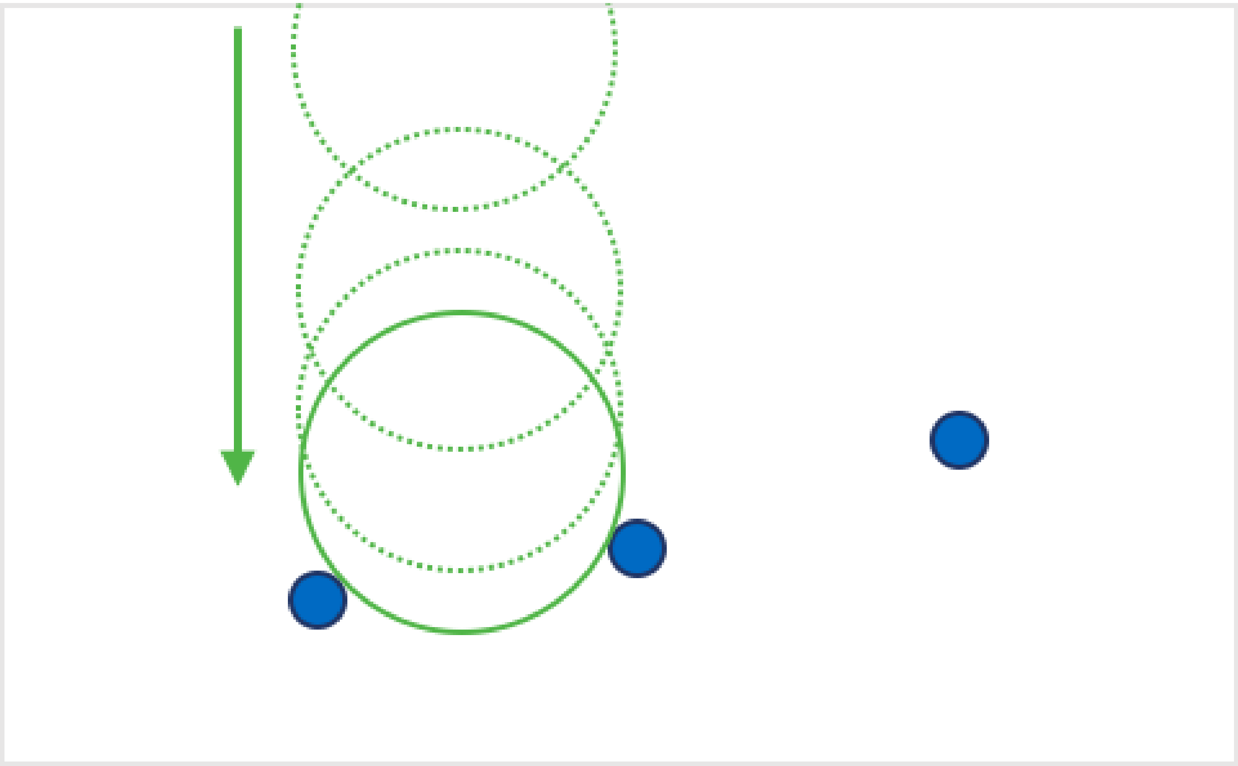
- Topographical Maps
  - Guidance and Navigation Algorithms
    - Self-driving Cars
    - Drones
- Object Classification
  - Facial Recognition
  - Vehicle Recognition



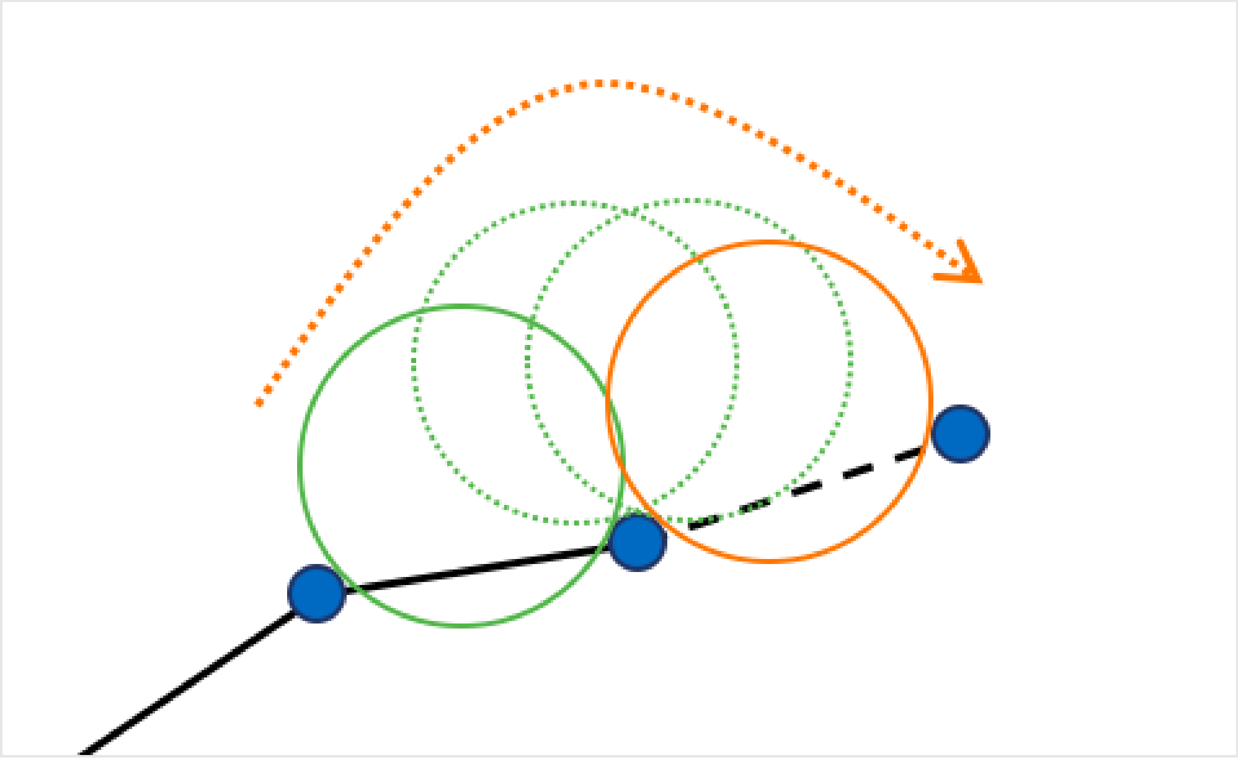
# Intuition Behind the Ball-Pivoting Algorithm



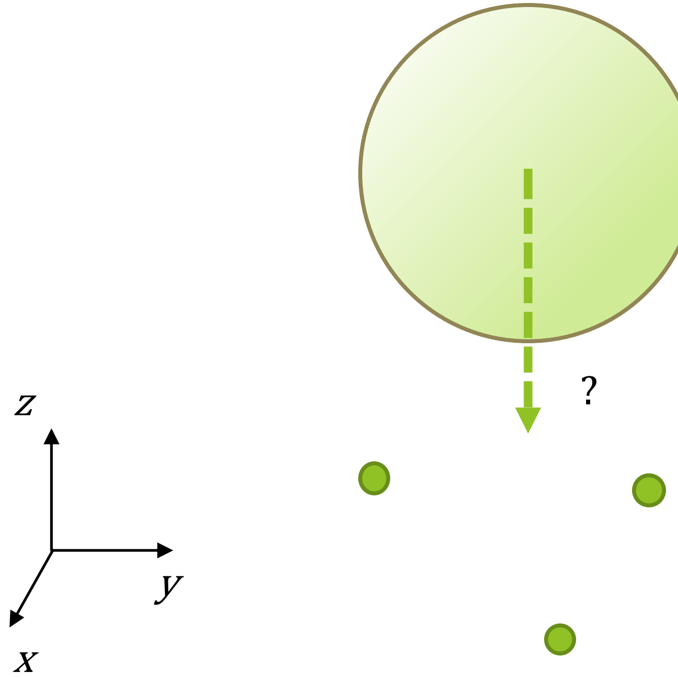
# Intuition Behind the Ball-Pivoting Algorithm



# Intuition Behind the Ball-Pivoting Algorithm



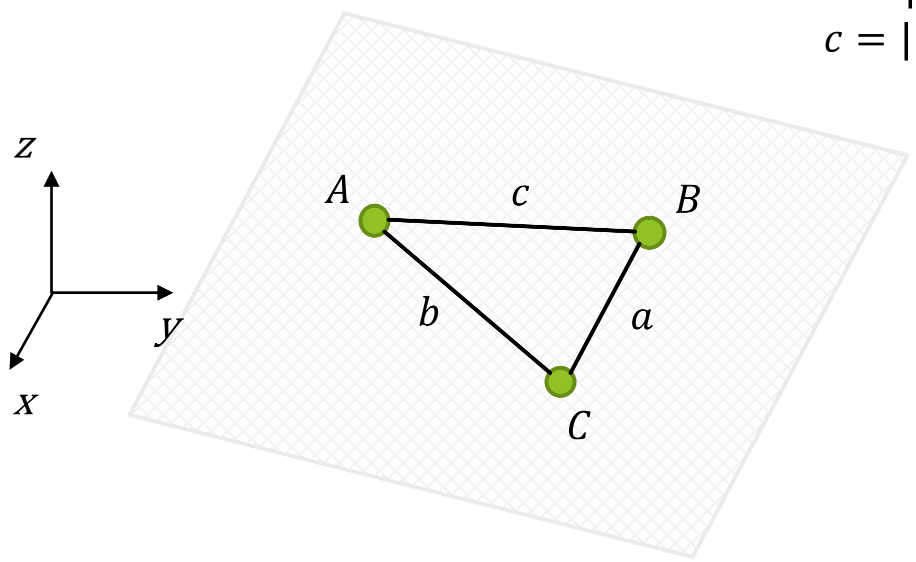
# Math: Setting a Ball Right-side-up on 3 Points



# Math: Setting a Ball Right-side-up on 3 Points

$$A = (x_a, y_a, z_a)$$
$$B = (x_b, y_b, z_b)$$
$$C = (x_c, y_c, z_c)$$

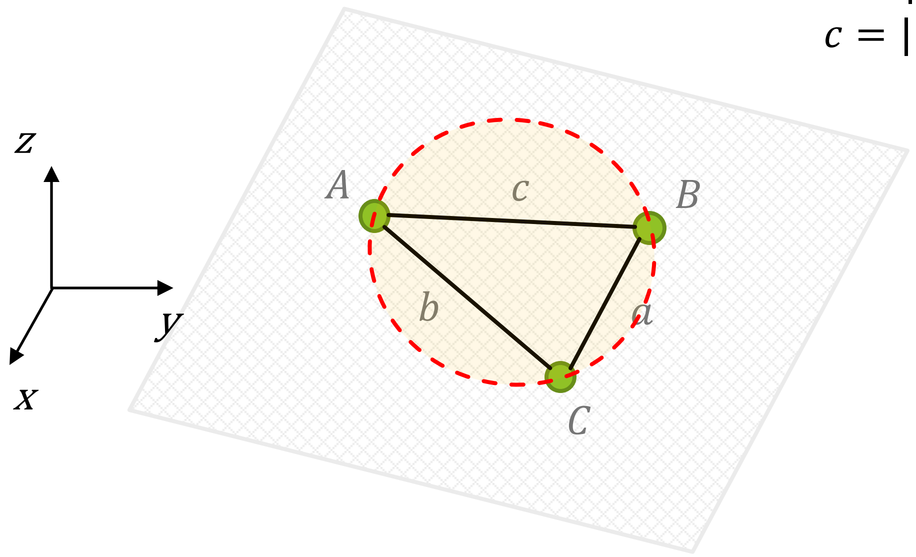
$$a = ||B - C||$$
$$b = ||A - C||$$
$$c = ||A - B||$$



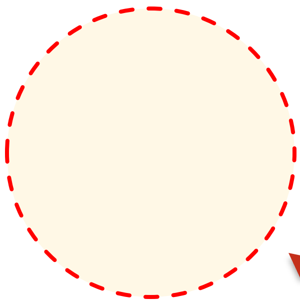
# Math: Setting a Ball Right-side-up on 3 Points

$$A = (x_a, y_a, z_a)$$
$$B = (x_b, y_b, z_b)$$
$$C = (x_c, y_c, z_c)$$

$$a = ||B - C||$$
$$b = ||A - C||$$
$$c = ||A - B||$$

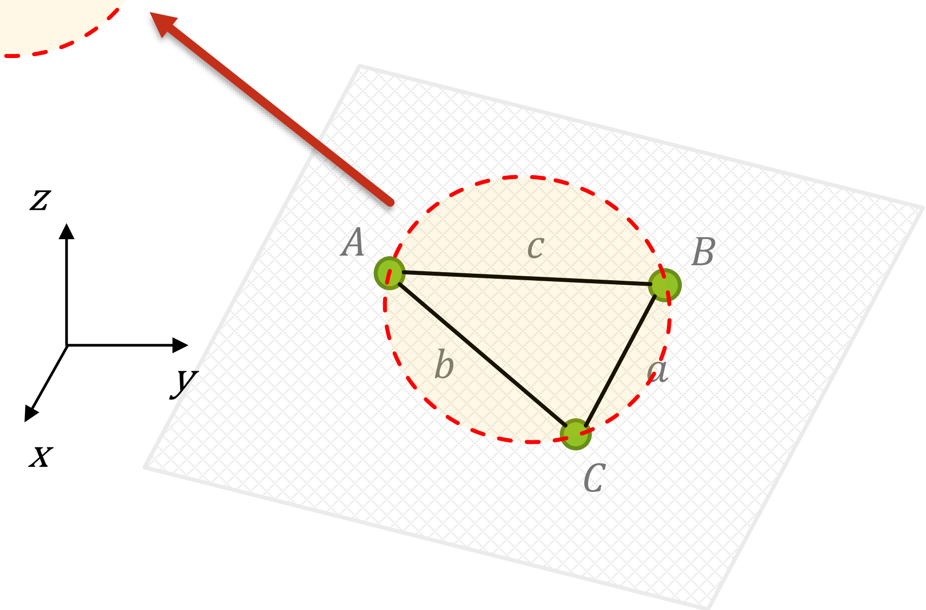


# Math: Setting a Ball Right-side-up on 3 Points



radius of 2D circumcenter [3]

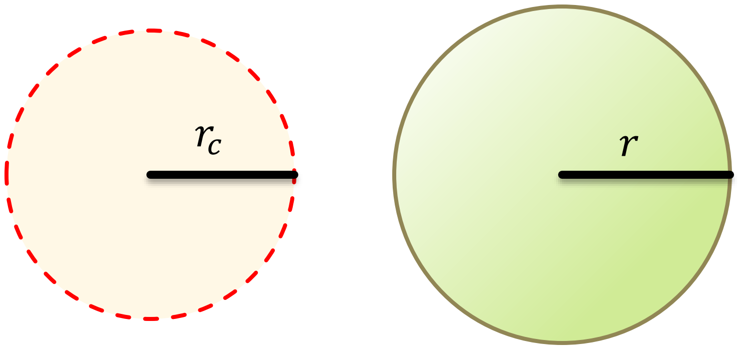
$$r_c = \sqrt{\left[\frac{a^2 b^2 c^2}{(a+b+c)(a+b-c)(b+c-a)(c+a-b)}\right]}$$




$$A = (x_a, y_a, z_a)$$
$$B = (x_b, y_b, z_b)$$
$$C = (x_c, y_c, z_c)$$

$$a = ||B - C||$$
$$b = ||A - C||$$
$$c = ||A - B||$$

# Math: Setting a Ball Right-side-up on 3 Points




$$A = (x_a, y_a, z_a)$$
$$B = (x_b, y_b, z_b)$$
$$C = (x_c, y_c, z_c)$$

$$a = ||B - C||$$
$$b = ||A - C||$$
$$c = ||A - B||$$

If $r_c \leq r$, the ball is able to touch all 3 points. We have a triangle!

If $r_c > r$, the ball can touch at most 2 points at a time, and "falls through." No triangle.

$$r_c = \sqrt{\left[\frac{a^2 b^2 c^2}{(a+b+c)(a+b-c)(b+c-a)(c+a-b)}\right]}$$

We can compare $r_c^2, r^2$ to be more efficient.

# Math: Setting a Ball Right-side-up on 3 Points

Find the “up” side of the triangle

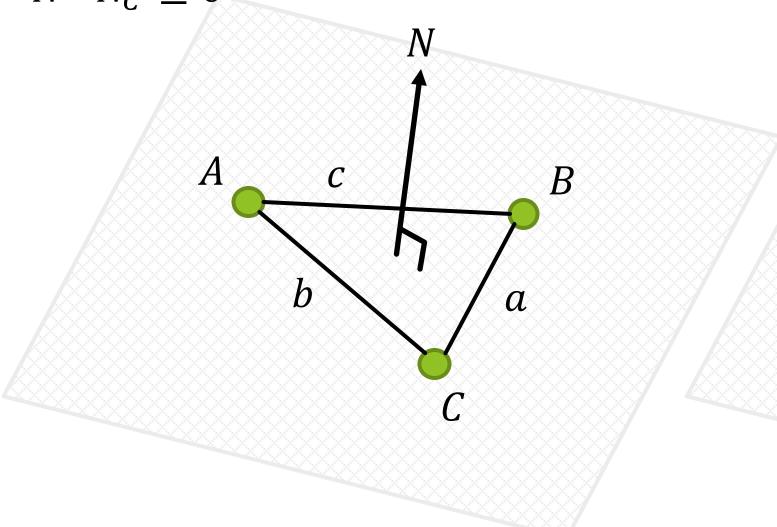
$$N = \frac{(B - A) \times (C - A)}{||(B - A) \times (C - A)||}$$

Check that: Otherwise, flip sign.

$N \cdot N_A \geq 0$

$N \cdot N_B \geq 0$

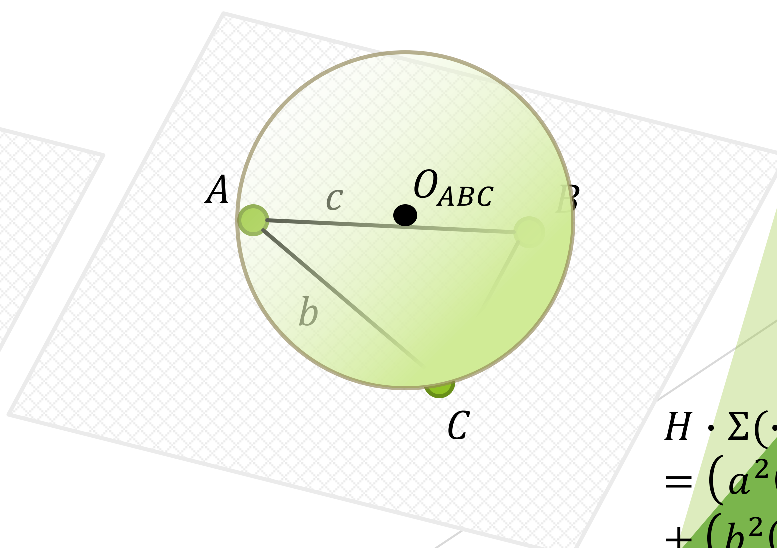
$N \cdot N_C \geq 0$




3D circumcenter given by

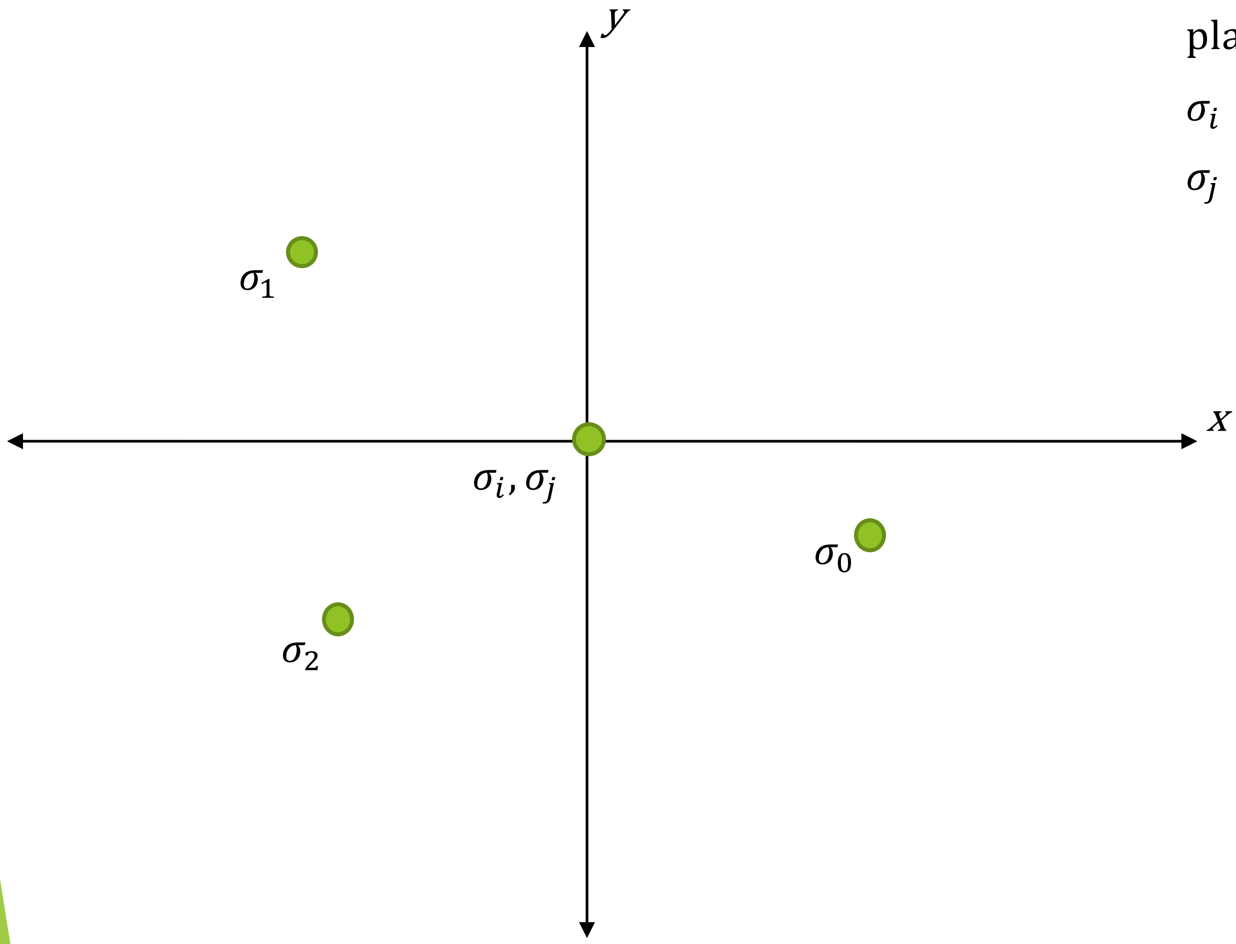
$$O = H + \sqrt{r^2 - r_c^2} \cdot N$$

Where $H$ is the location of the 2D circumcenter [3]




$$\begin{aligned} H \cdot \Sigma(\cdot) &= \left(a^2(b^2 + c^2 - a^2)\right)A \\ &+ \left(b^2(a^2 + c^2 - b^2)\right)B \\ &+ \left(c^2(a^2 + b^2 - c^2)\right)C \end{aligned}$$

# Ball Rolling Math

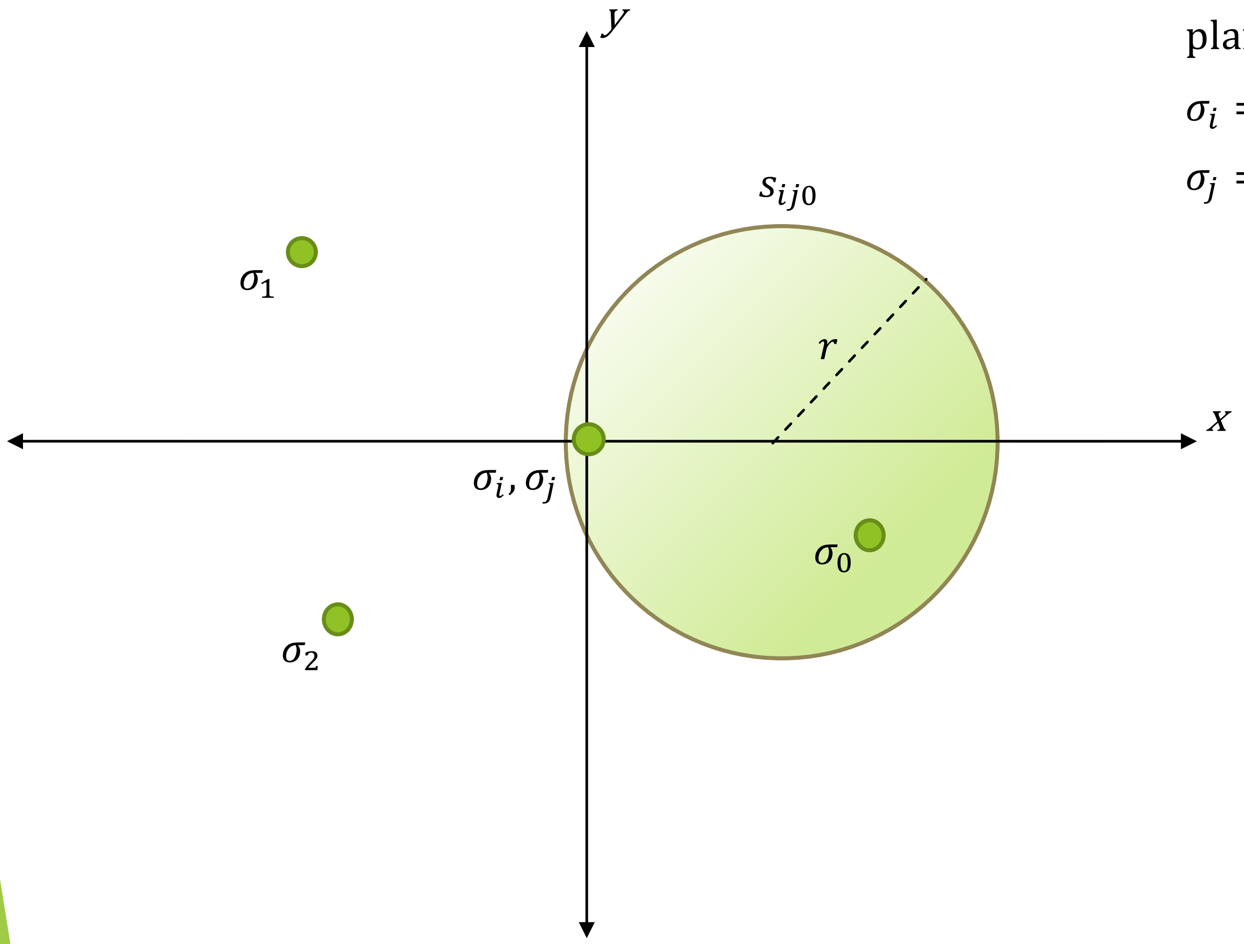



plane $z = 0$

$\sigma_i = (0,0,+a)$

$\sigma_j = (0,0,-a)$

# Ball Rolling Math

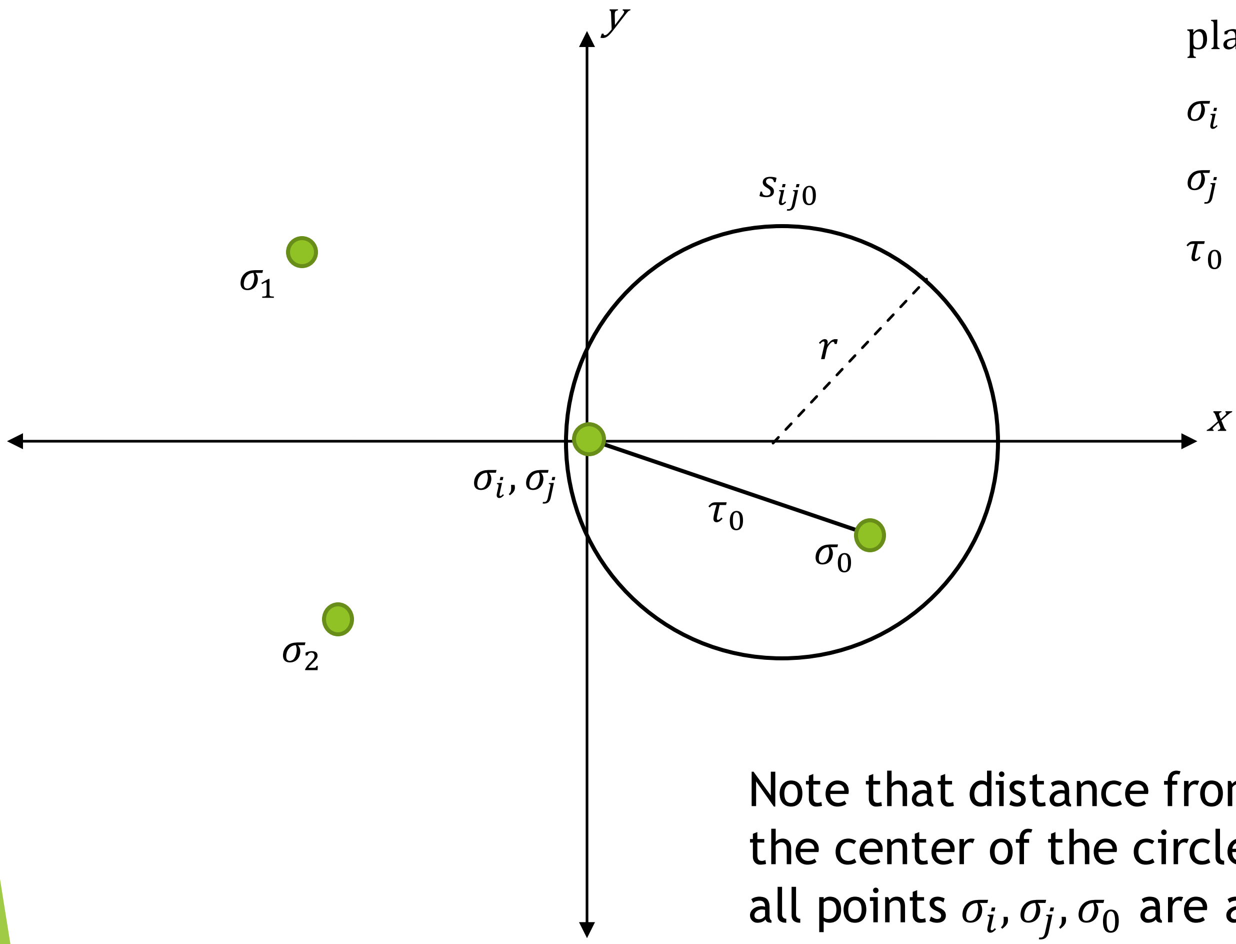



plane $z = 0$

$\sigma_i = (0,0,+a)$

$\sigma_j = (0,0,-a)$

# Ball Rolling Math




plane $z = 0$

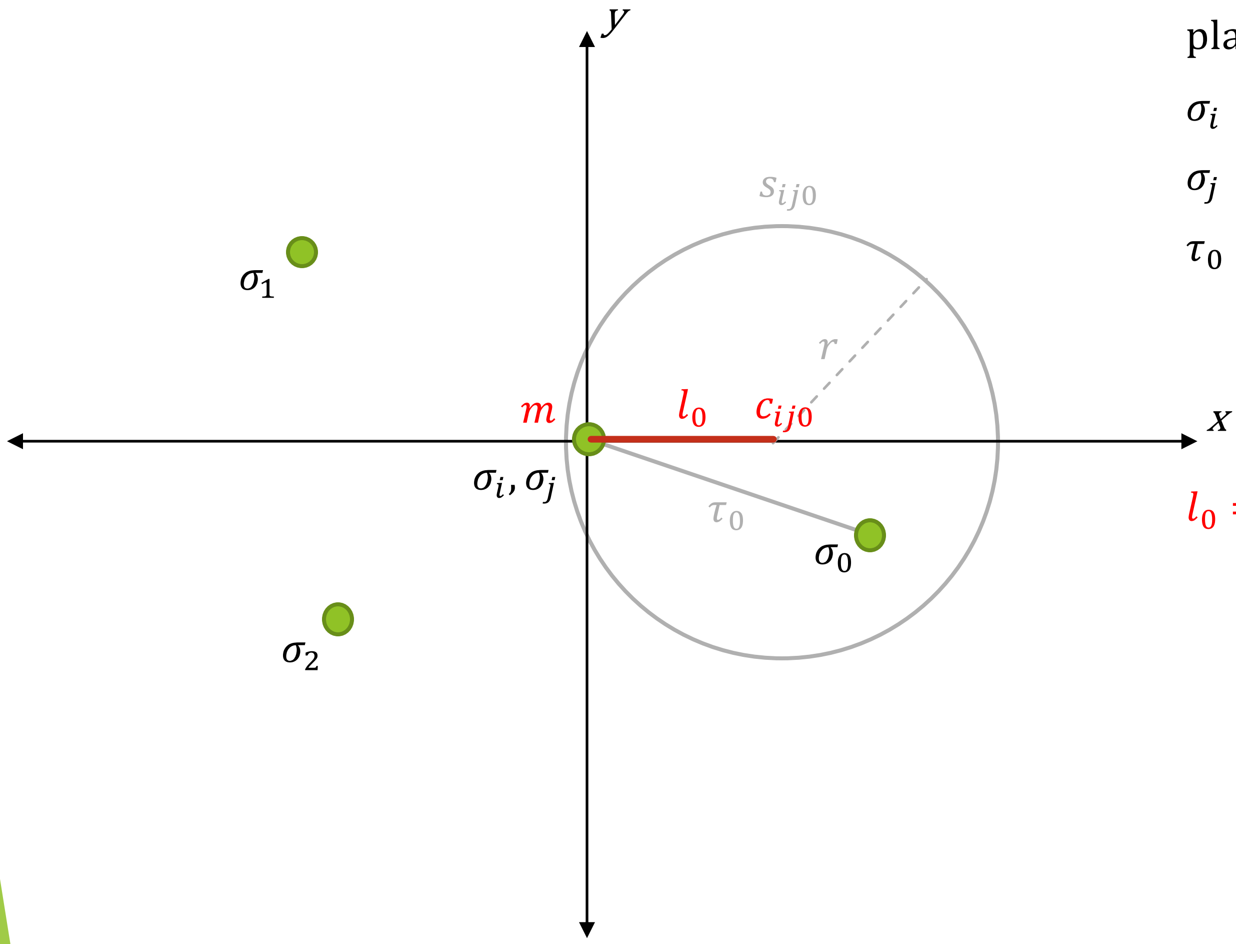
$\sigma_i = (0,0,+a)$

$\sigma_j = (0,0,-a)$

$\tau_0 = (\sigma_i, \sigma_j, \sigma_0)$

Note that distance from the center of the circle to all points $\sigma_i, \sigma_j, \sigma_0$ are all $r$.

# Ball Rolling Math




plane $z = 0$

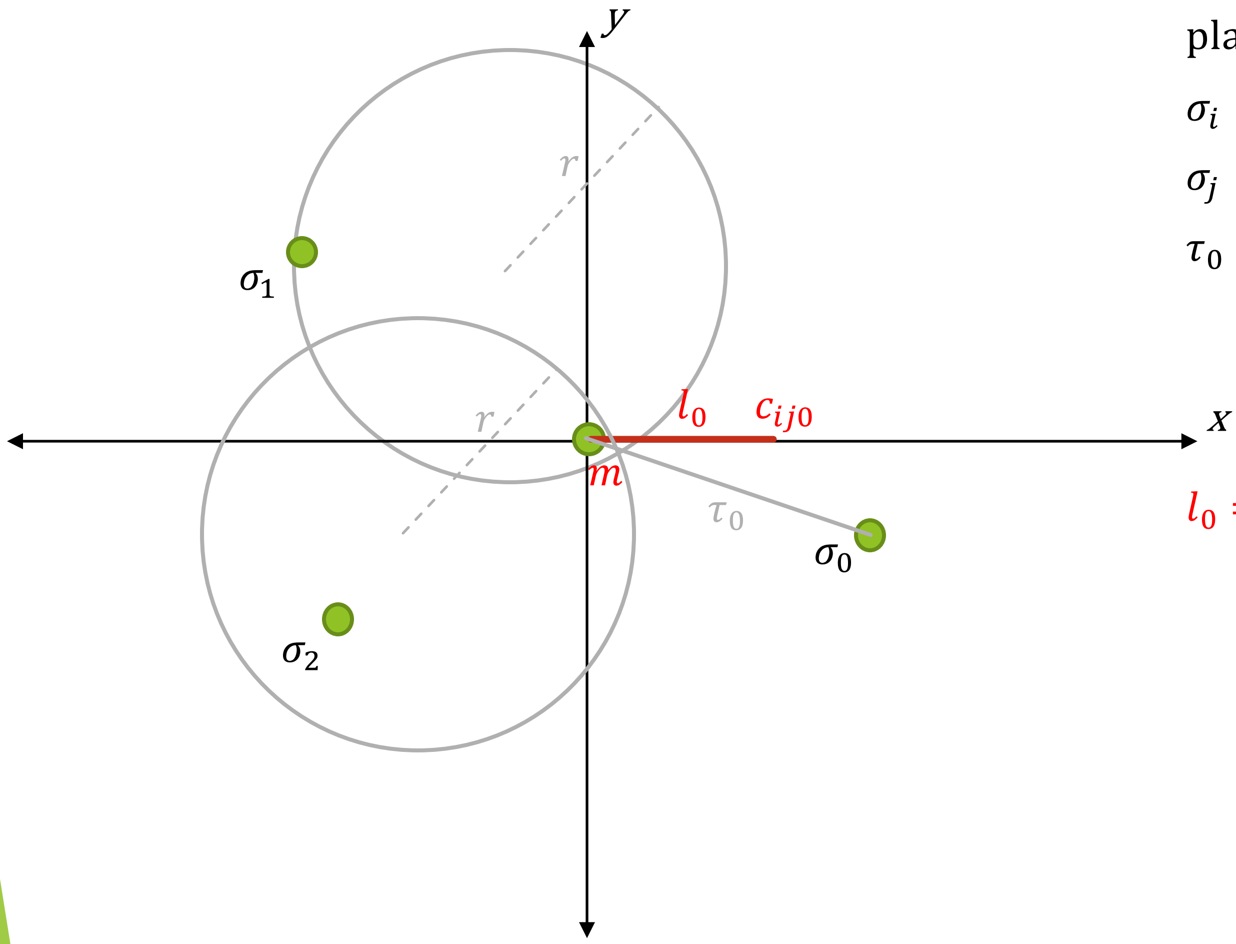
$\sigma_i = (0,0,+a)$

$\sigma_j = (0,0,-a)$

$\tau_0 = (\sigma_i, \sigma_j, \sigma_0)$

$l_0 = c_{ij0} - m$

# Ball Rolling Math




plane $z = 0$

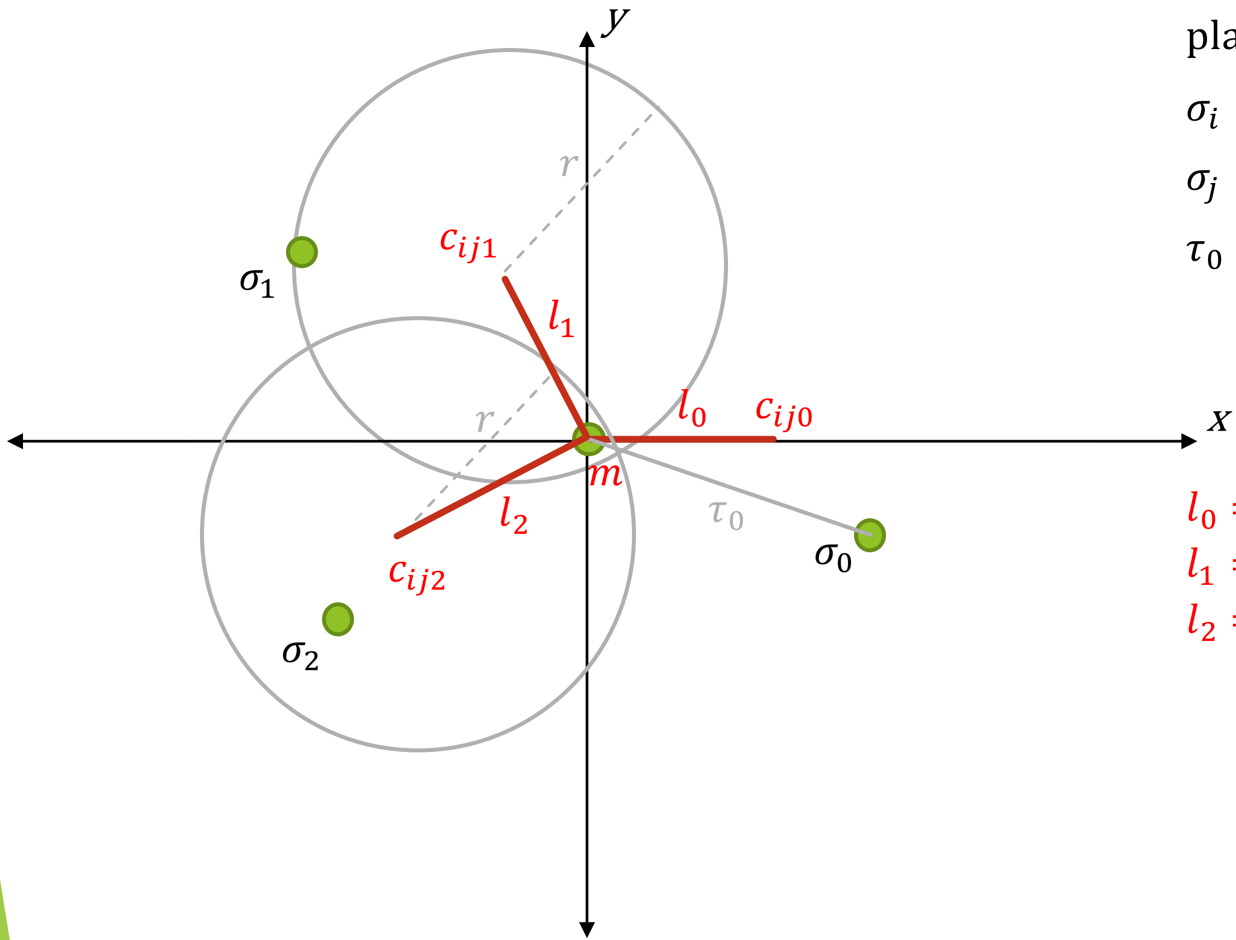
$\sigma_i = (0,0,+a)$

$\sigma_j = (0,0,-a)$

$\tau_0 = (\sigma_i, \sigma_j, \sigma_0)$

$l_0 = c_{ij0} - m$

# Ball Rolling Math




plane $z = 0$

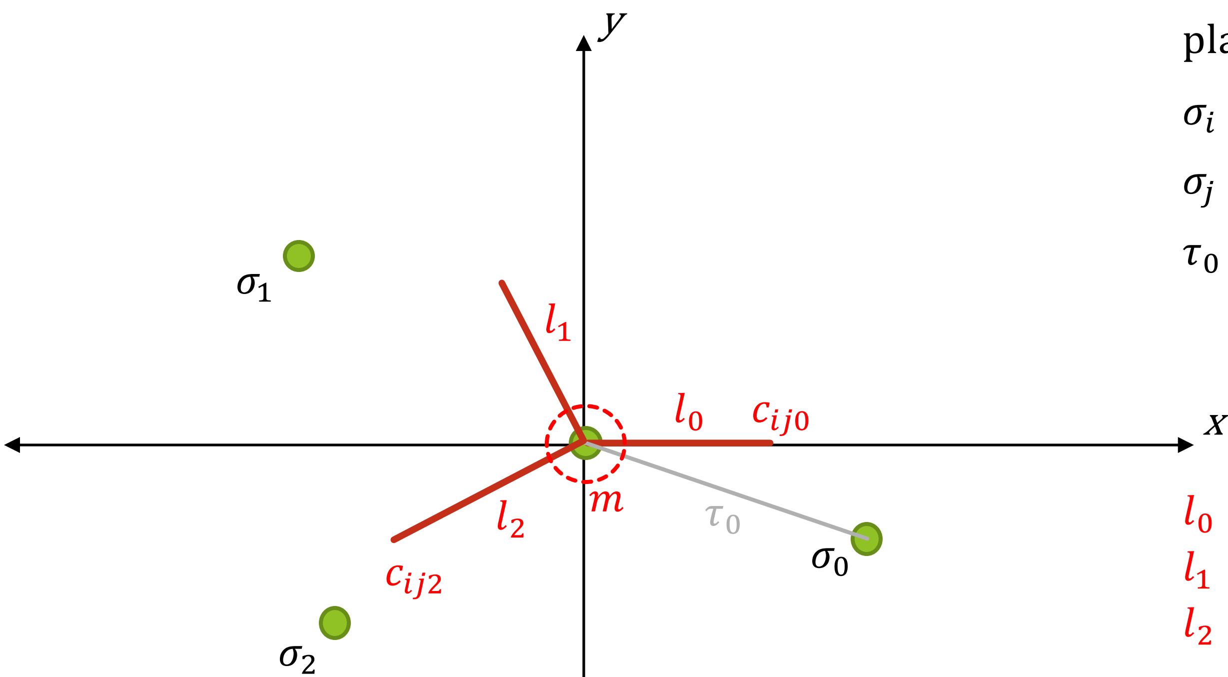
$\sigma_i = (0,0,+a)$

$\sigma_j = (0,0,-a)$

$\tau_0 = (\sigma_i, \sigma_j, \sigma_0)$

$l_0 = c_{ij0} - m$
$l_1 = c_{ij1} - m$
$l_2 = c_{ij2} - m$

# Ball Rolling Math




plane $z = 0$

$\sigma_i = (0,0,+a)$

$\sigma_j = (0,0,-a)$

$\tau_0 = (\sigma_i, \sigma_j, \sigma_0)$

$l_0 = c_{ij0} - m$
$l_1 = c_{ij1} - m$
$l_2 = c_{ij2} - m$

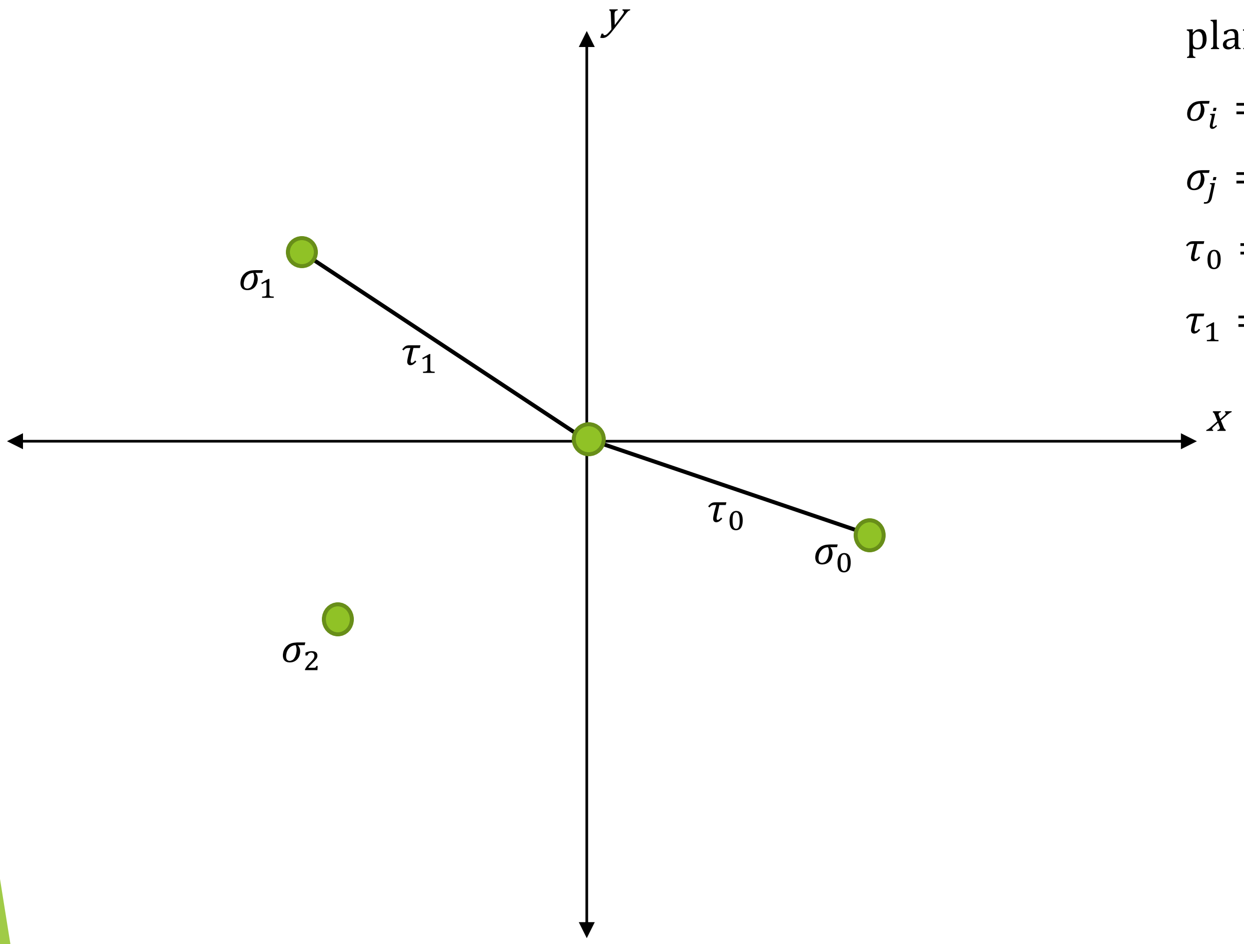
Find the new line segment with the smallest angle compared to the initial segment.
(All line segments lie on the same plane.)

$\measuredangle(l_1, l_0); \measuredangle(l_2, l_0); \ldots$

$$\measuredangle(l_n, l_0) = \theta = \text{acos}\left(\frac{l_n \cdot l_0}{||l_n \cdot l_0||}\right)$$

$\theta \rightarrow 2\pi - \theta$ if $(l_0 \times l_n) \cdot \left(\sigma_i - \sigma_j\right) < 0$

# Ball Rolling Math




plane $z = 0$

$\sigma_i = (0,0,+a)$

$\sigma_j = (0,0,-a)$

$\tau_0 = (\sigma_i, \sigma_j, \sigma_0)$

$\tau_1 = (\sigma_i, \sigma_j, \sigma_1)$

# Ball-Pivoting Algorithm Overview

- Inputs: Point cloud with point normals, ball radius
- Outputs: A list of triangles representing a surface

1. Preprocessing & Creating data Structures
2. While (There are unvisited points)
3. <u>**Find Seed Triangle**</u>, add triangle to output
4. <u>**Expand Triangulation**</u>, add triangles to output
5. Rerun with larger radii & Postprocessing

# Find Seed Triangle

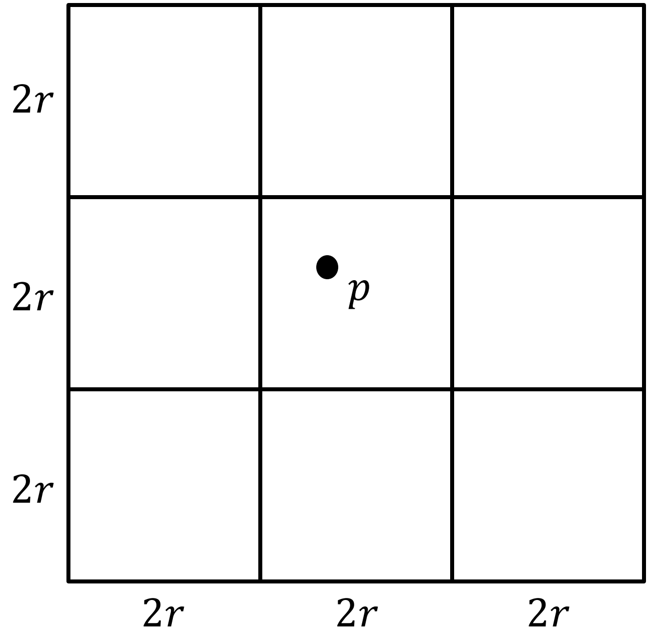
- Keep track of which points have been visited
- Pick an unvisited point *p*
- Look in the 27-neighborhood of the point, using the **Voxel Grid**
- Sort all neighboring points on increasing distance
- Find the first triplet of points (*p*,*q*,*s*) that:
  - Their triangle normal has positive scalar products with all point normals
  - The *r*-ball placed on those three points have NO other points inside the ball. "Empty ball configuration." Check that no other point is within *r* of the circumcenter
- Add new triangle to list of final output triangles

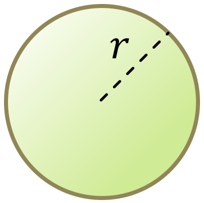
# Expand Triangulation

- Add the seed triangle's 3 edges to an empty edge container named "expansion front"
- While (expansion front is not empty)
- Pop an edge *e* from the expansion front
- Skip *e* if it is a boundary or inner edge
- "Roll" the ball, mathematically find new point *p*, verify point
  - If no point is found, mark *e* as a boundary edge and go to next iteration
- Add new triangle to list of final output triangles
- Two edges show up ($p$, $e.a$) and ($p$, $e.b$). For each edge
  - If edge touches two triangles, mark edge as an inner edge
  - Else push edge to expansion front

# Voxel Grid Data Structure

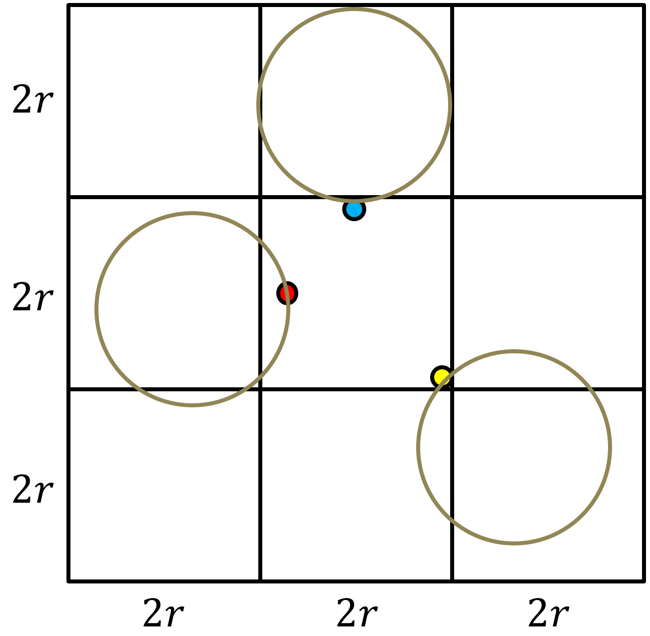
- Points can be retrieved by indexing
- Used for looking up neighboring points
- Each voxel is $2r \times 2r \times 2r$ sized
- Each point belongs to a voxel
- Each voxel has 27 neighbors, including itself

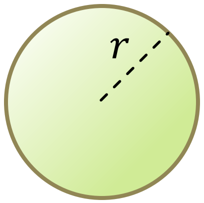


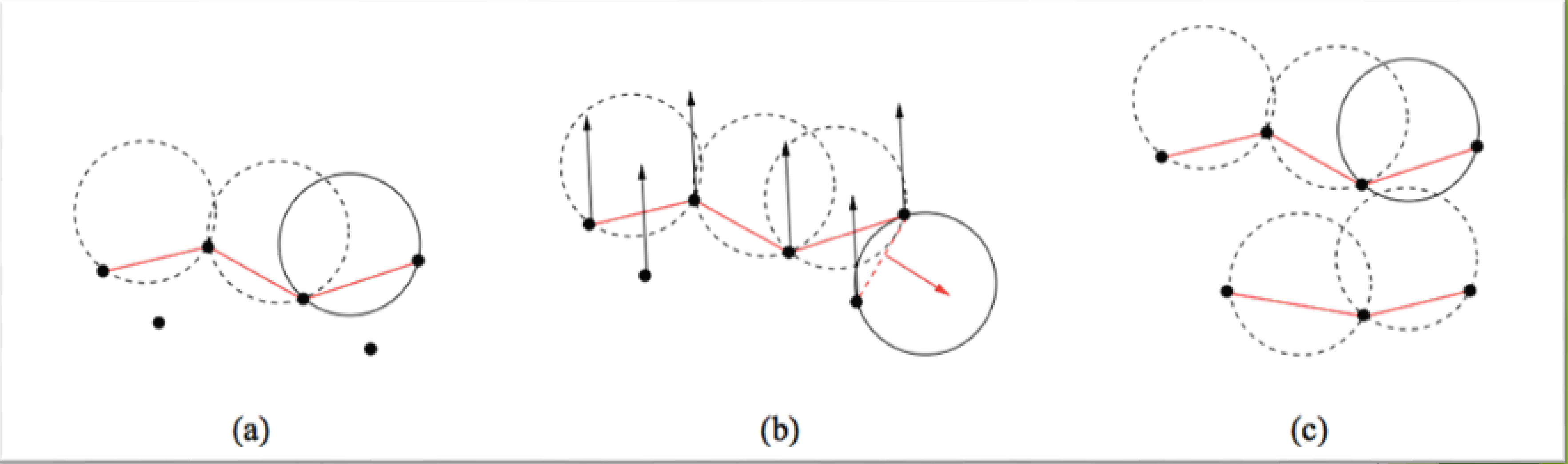
# Voxel Grid Data Structure

- Points can be retrieved by indexing
- Used for looking up neighboring points
- Each voxel is $2r \times 2r \times 2r$ sized
- Each point belongs to a voxel
- Each voxel has 27 neighbors, including itself



# Edge Case: Noisy Data




- (a) Samples below the surface are ignored
- (b) Triangle normals are checked against point normals so that the surface is consistently oriented. Pivoting edge becomes a border edge.
- (c) Two surfaces are created. The small error surface can be removed in post-processing

# Edge Case: Irregular Holes in the Surface

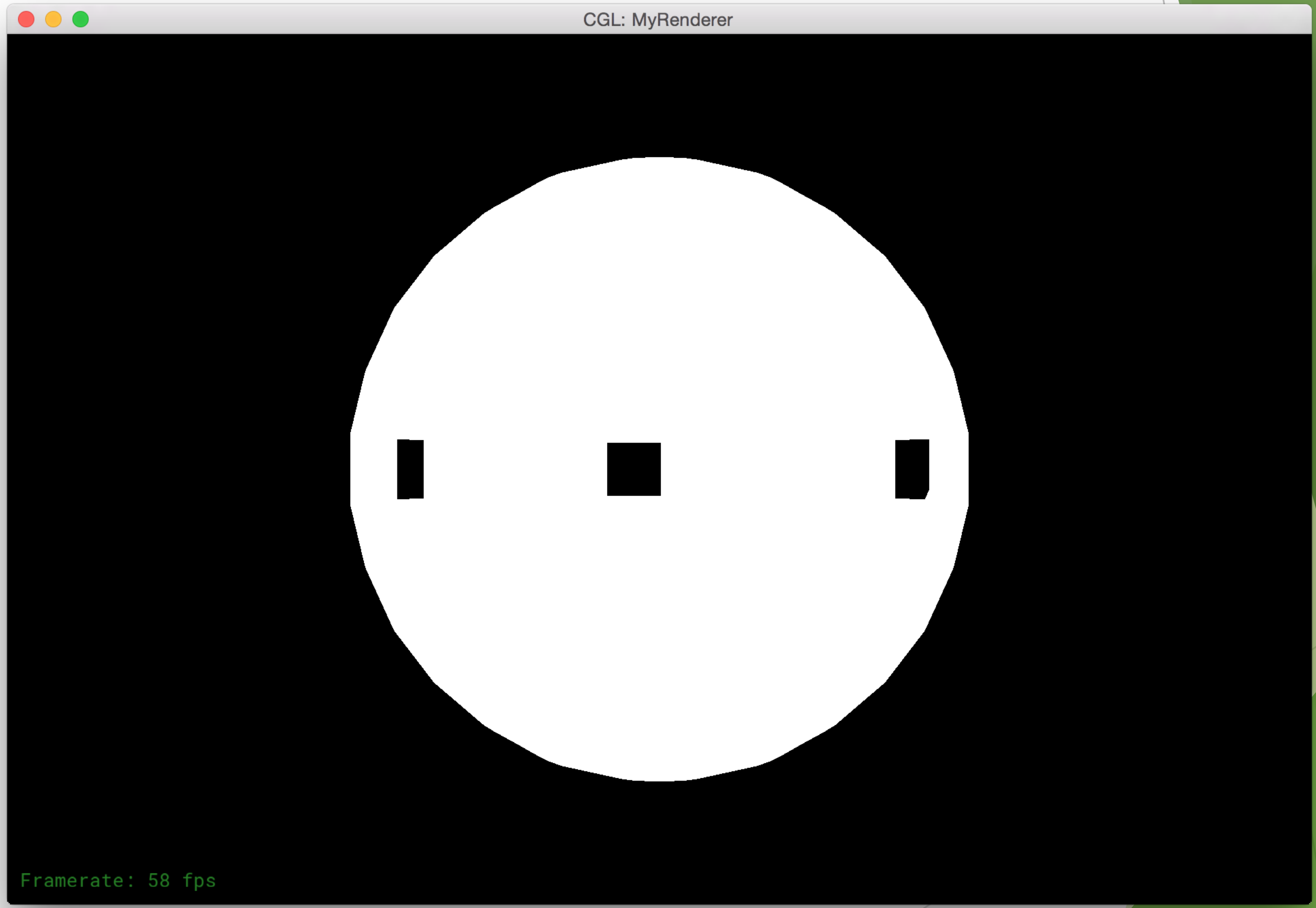
(a) Detail loss

(b) Hole creation

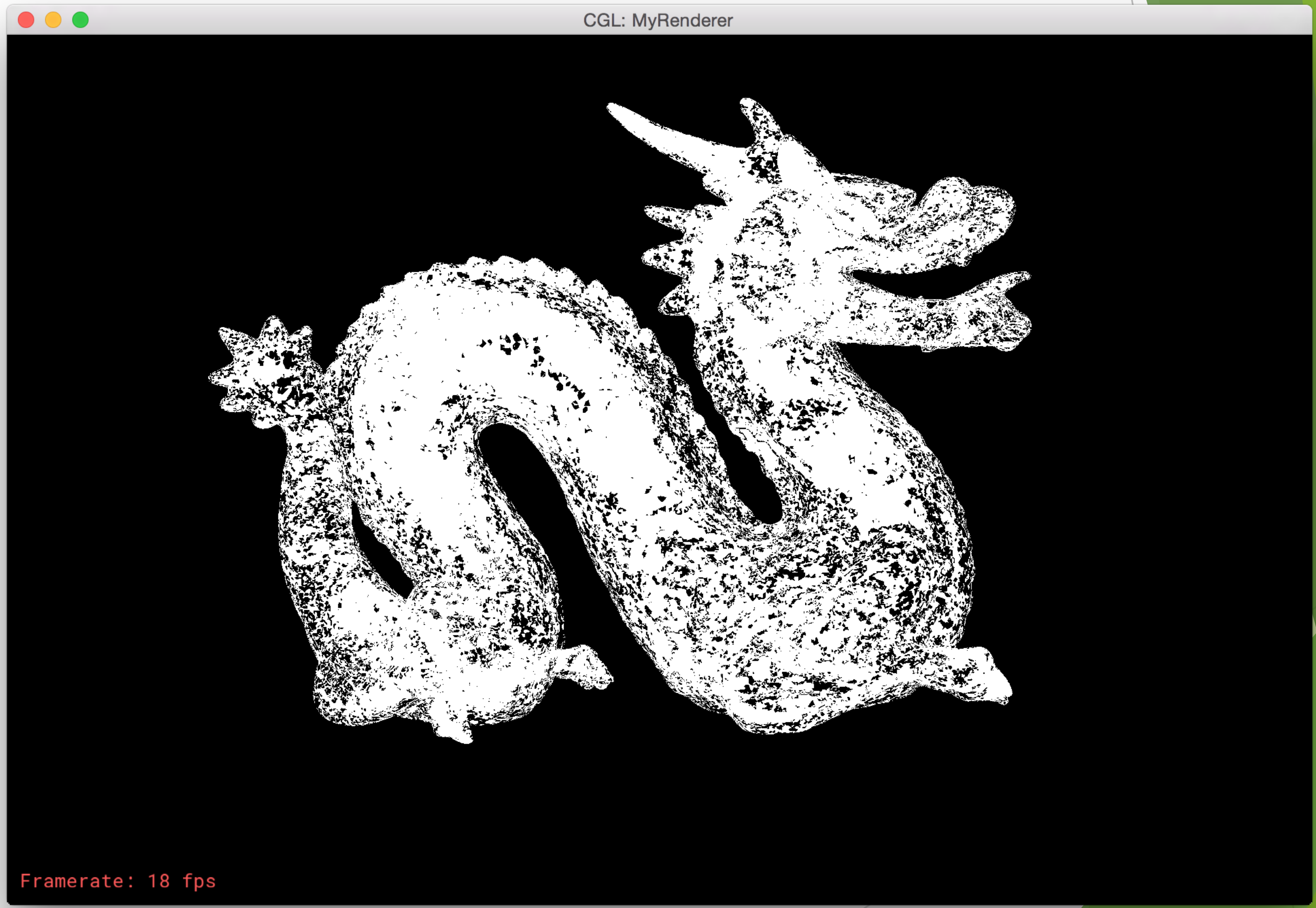
(c) A possible correction: using multiple radii

- (a) Ball radius is too large
- (b) Ball radius is too small (and too large for bottom point
- (c) Run the BPA using multiple radii, in increasing order

# Test Results



# Real Data Results



# Real Data Results

- https://www.youtube.com/watch?v=CevV4jS45Bc

# Limitations

- Algorithm slows down as radii increases relative to data
- Limited by finding new triangle operation
  - Looking in 27-neighborhood
  - For each point in increasing distance, check if the triangle it makes is valid
  - For all reachable points, find the one with minimum angle
- Radii chosen empirically, good starting size is average distance from any point to closest neighbor
- Small radii lead to holes, large radii lead to detail loss

# Possible Speed Improvements

- Parallelizable – Have multiple balls rolling at once
- Use an octree to support parallelism, each ball runs in it’s own octant
- Scale radius based on point density of 3D space
  - Requires an octree structure which keeps track of data statistics
  - May misinterpret noisy data

# Conclusion

- The quality of the reconstructed surface heavily depends on the choice of the radii
- Algorithm speed decreases with radii increase
- Only obstacle to a fully automatic algorithm is the radius selection. There are naive strategies to select radii.
- Works with noisy data, but works best with smooth and uniformly sampled data
- Can be sped up by parallelization and advanced data structures

# References


- F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, *The Ball-Pivoting Algorithm for Surface Reconstruction*, http://www.research.ibm.com/vistechnology/pdf/bpa_tvcg.pdf
- Julie Digne, *An Analysis and Implementation of a Parallel Ball Pivoting Algorithm*, http://www.ipol.im/pub/art/2014/81/article.pdf
- Wolfram MathWorld, *Circumradius*, http://mathworld.wolfram.com/Circumradius.html