CSCI 4560/6560 Computational Geometry

https://www.cs.rpi.edu/~cutler/classes/computationalgeometry/F23/

# Lecture 20: Binary Space Partitions

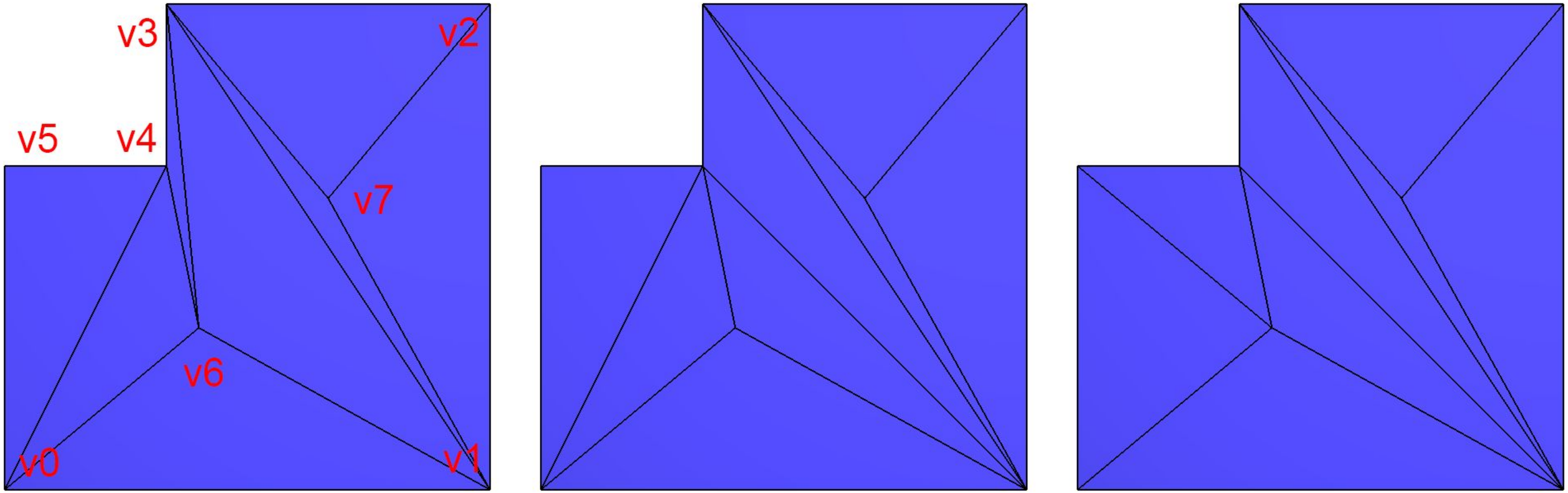
# Outline for Today

- Homework 7 & Final Project Proposal Feedback
- Last Time: General Position, Robustness, & Exact Computation
- Line Drawings & Early Computer Vision / AI
- Hidden Line Drawing: z-Buffer
- Hidden Line Drawing: Painter's Algorithm
- Binary Space Partition
- Binary Space Partition Analysis
- Discussion & Comparison to Quad Tree & kD Tree
- Final Project 3D Visualization Challenges
- Next Time: Polyominoes & Tiling

# Homework 7

- Ok if your solution is *not the shortest path*
  *(e.g., it has unnecessary edits that are later reverted)*

```
v3-v6  ->  v1-v4
v0-v4  ->  v5-v6
v1-v3  ->  v4-v7
v1-v4  ->  v6-v7
v6-v7  ->  v1-v4
v5-v6  ->  v0-v4
v3-v7  ->  v2-v4
```




...

# Final Project Proposals & Progress Post #1

- *If you haven't submitted your proposal yet, please do so ASAP*
- I've graded the Final Project Proposals, Common feedback includes:
  - Missing a title!
  - Who is your audience? Your classmates! Describe technical details as appropriate (prereqs & technical content covered in lecture/hw)
  - Project scope is vague / project scope is likely too large
  - Didn't describe a specific set of examples / sample data that will allow you to debug your work and prepare for presentation & report
  - Didn't include full bibliography citations, didn't use a standard callout within document to the bibliography, e.g., "[1]" or "(Smith 2010)"
- If you would like to revise & re-submit your proposal you can do that…
  *Or just take the feedback and use it when you write your final project report*
- Progress Post #1 due on Monday Nov 13th on Submitty forum

# Outline for Today

- Homework 7 & Final Project Proposal Feedback
- Last Time: General Position, Robustness, & Exact Computation
- Line Drawings & Early Computer Vision / AI
- Hidden Line Drawing: z-Buffer
- Hidden Line Drawing: Painter's Algorithm
- Binary Space Partition
- Binary Space Partition Analysis
- Discussion & Comparison to Quad Tree & kD Tree
- Final Project 3D Visualization Challenges
- Next Time: Polyominoes & Tiling

# Factorization by Gaussian Elimination

- Divide by zero is not the only concern…
- We should also avoid division by very small values, e.g., epsilon:

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{A} = \begin{bmatrix} -\epsilon & 1 \\ 1 & -1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1-\epsilon \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} -\epsilon & 1 & 1-\epsilon \\ 0 & -1+\epsilon^{-1} & \epsilon^{-1}-1 \end{bmatrix} \Rightarrow \begin{aligned} x_2 &= 1 \\ x_1 &= \frac{(1-\epsilon)-1}{-\epsilon} \end{aligned}$$

*Correct answer: $x_1$=1*
*But we will have robustness problems if ε is very small!*

*It's better to pivot / swap rows for the row with the largest value in this column*

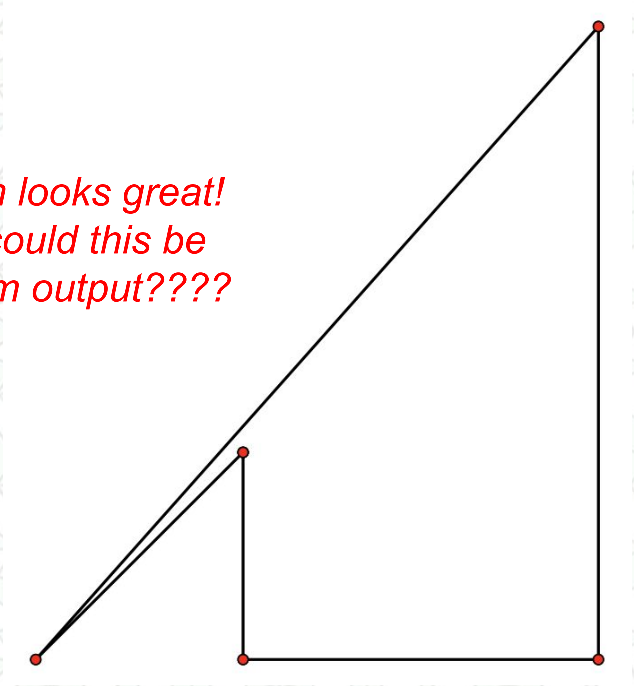
Fundamentals of Numerical Computation, Driscoll & Braun, 2017
https://fncbook.github.io/v1.0/linsys/pivoting.html

# Incremental Convex Hull Construction

- Make a triangle with the first 3 points

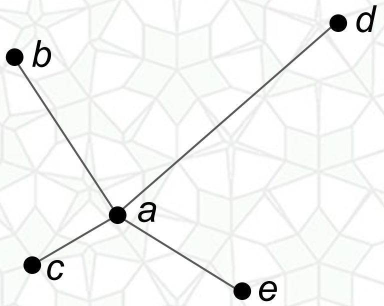
- For each additional point *r*
  - Find an outside edge that is "visible" from *r*
  - Expand to a sequence of connected edges $v_i \rightarrow v_{i+1} \rightarrow v_{i+2} \; (\rightarrow \; \ldots \; ) \rightarrow v_j$
  - Remove middle points (e.g., $v_{i+1}$ & $v_{i+2}$) from hull, add point *r* to hull

*Algorithm looks great!*
*So how could this be*
*a program output????*



"Geometric Computing: The Science of Making Geometric Algorithms Work", Kurt Mehlhorn
https://people.mpi-inf.mpg.de/~mehlhorn/ftp/SoCG09.pdf

# Avoid Creating Irrational Numbers

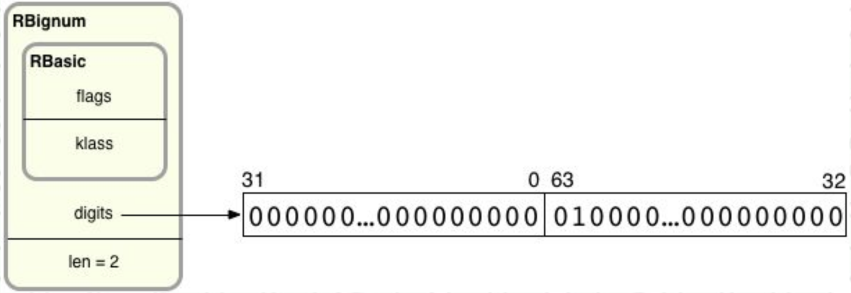


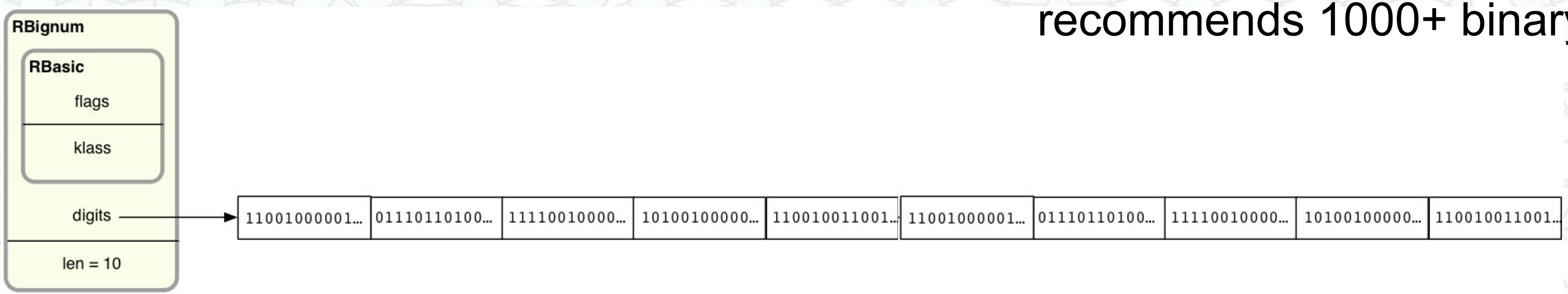

- Problem: Given 5 points with integer coordinates, find the nearest neighbor to point *a*
- Compute the length of lines *ab, ac, ad, ae*
  - length(*ab*) = sqrt ( $(x_a - x_b)*(x_a - x_b) + (y_a - y_b)*(y_a - y_b)$ )
  - Note: the sqrt, will likely create irrational numbers!
- Sort the lengths, return endpoint for shortest line length

- Instead… compute & sort the squares of the line lengths
  - squared_length(*ab*) = $(x_a - x_b)*(x_a - x_b) + (y_a - y_b)*(y_a - y_b)$
  - This is an integer!
- This will always return the correct answer to the original question. *WITHOUT creating irrational numbers!*

# Arbitrary Precision Arithmetic

- If we do not have irrational numbers in our program…
- We can store integers using a “BigNum” infinite precision integer type




- 64 bit binary integer =
  ~19 bit base 10 integer

- RSA Security requires at least 100 binary digits, but recommends 1000+ binary digits

https://patshaughnessy.net/2014/1/9/how-big-is-a-bignum

# Arbitrary Precision Arithmetic

- If we do not have irrational numbers in our program…
- We can store rational numbers as a ratio of two BigNums
- Reduce fractions whenever possible to minimize storage:

$$\frac{49578291287491495151508905425869578}{74367436931237242727263358138804367} \rightarrow \frac{2}{3}$$

https://algorist.com/problems/Arbitrary-Precision_Arithmetic.html

# Outline for Today

- Homework 7 & Final Project Proposal Feedback
- Last Time: General Position, Robustness, & Exact Computation
- Line Drawings & Early Computer Vision / AI
- Hidden Line Drawing: z-Buffer
- Hidden Line Drawing: Painter's Algorithm
- Binary Space Partition
- Binary Space Partition Analysis
- Discussion & Comparison to Quad Tree & kD Tree
- Final Project 3D Visualization Challenges
- Next Time: Polyominoes & Tiling

# Motivation: Summer Vision Project

- “Summer Vision Project”
  1966
  10 undergraduate students at MIT were tasked with solving computer vision

*It was a “BHAG”:*
*Big Hairy Audacious Goal*

*Did they (professor/students) realize it at the time???*

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

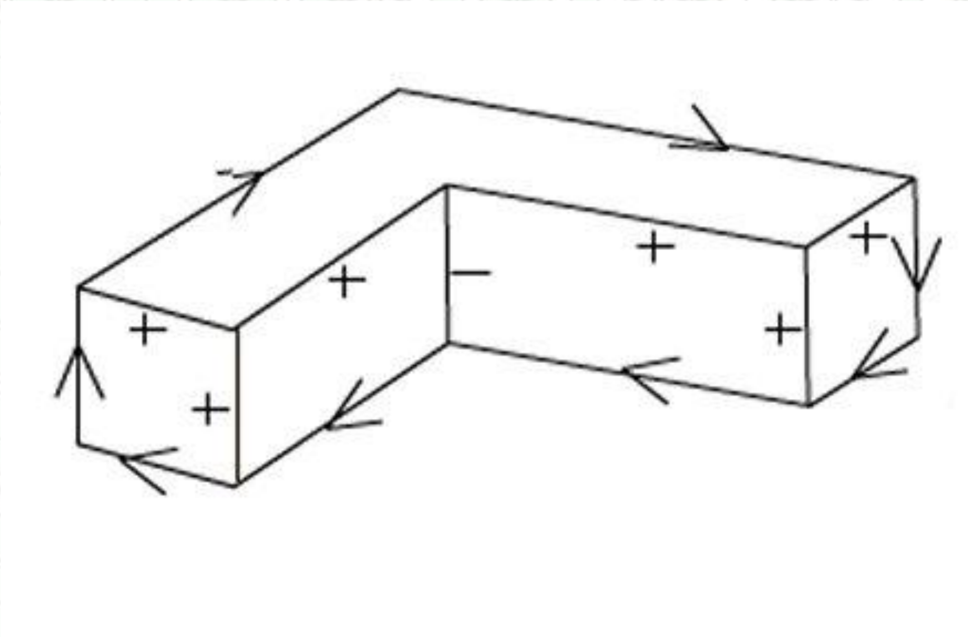
Artificial Intelligence Group July 7, 1966
Vision Memo. No. 100.

THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

# Motivation: Early AI & Early Computer Vision

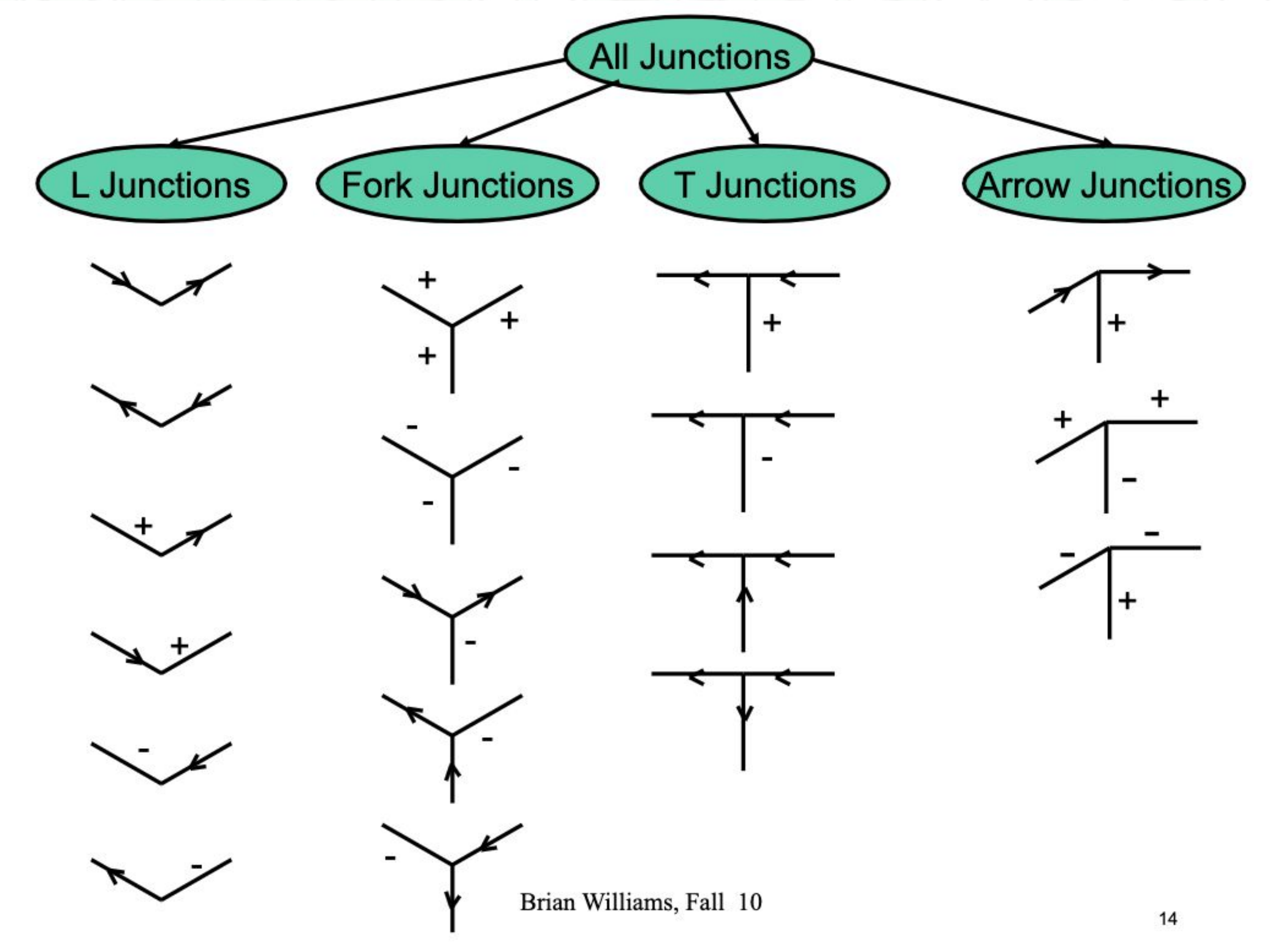



http://www-g.eng.cam.ac.uk/mmg/teaching/artificialintelligence/nonflash/constraint2.htm

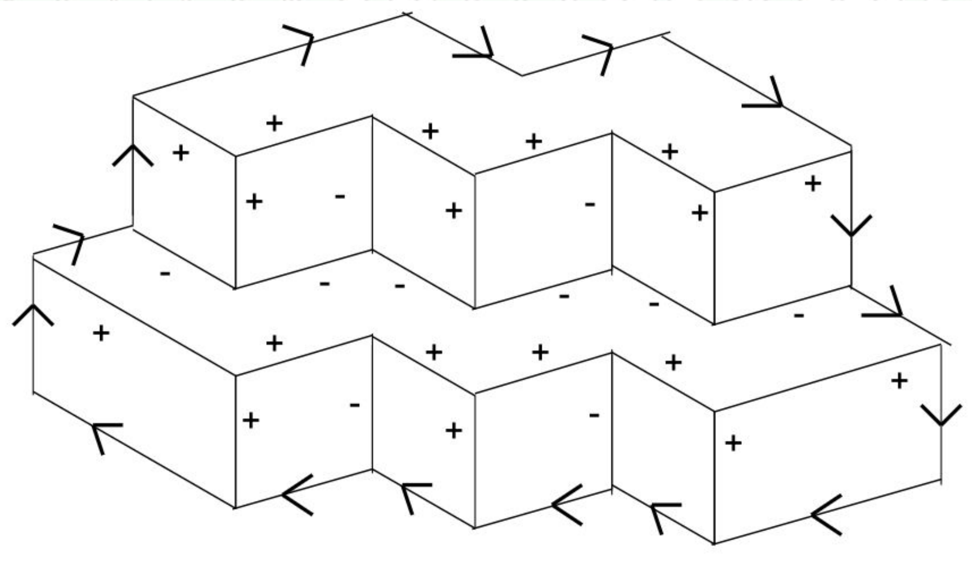
# Line Labeling Constraint Propagation

“Interpretation of opaque, trihedral solids with no surface marks”, Huffman & Clowes, 1971

“Compute Labeling through Local Propagation”
Waltz, 1972



# Motivation: Early AI & Early Computer Vision

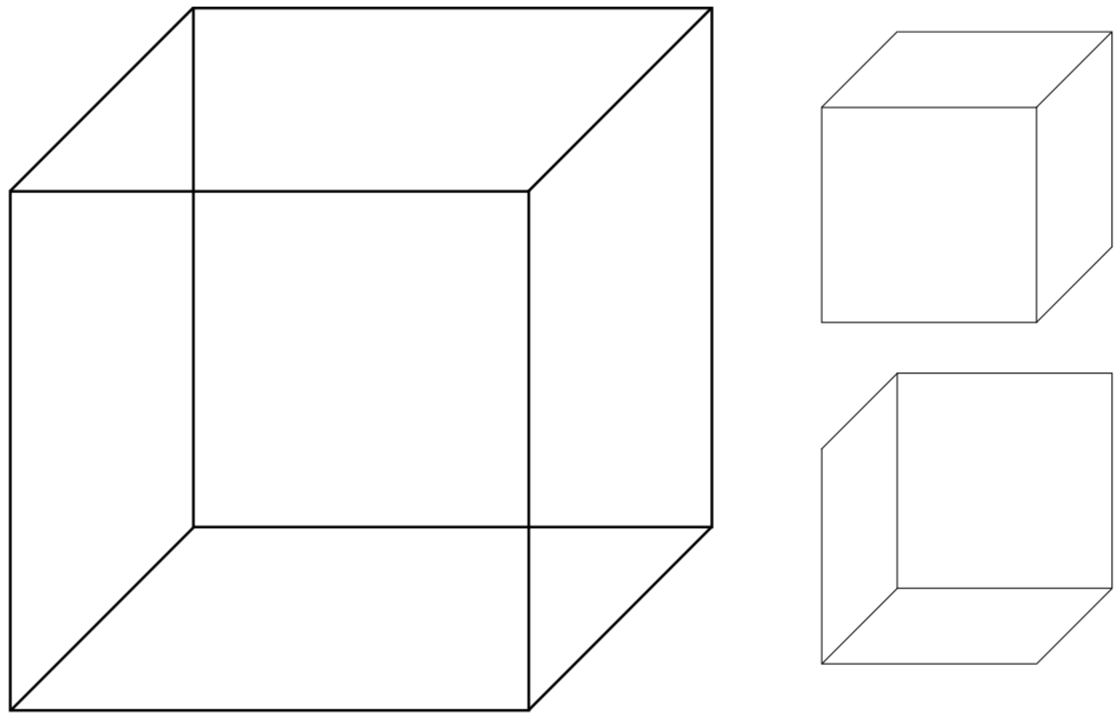


MIT 6.034 Artificial Intelligence, Fall 2010
Open CourseWare
https://www.youtube.com/watch?v=l-tzjenXrvI

# Necker Cube

- A two dimensional representation of a three dimensional wire frame cube

- Viewer's perception can flips back and forth between equally possible perspectives

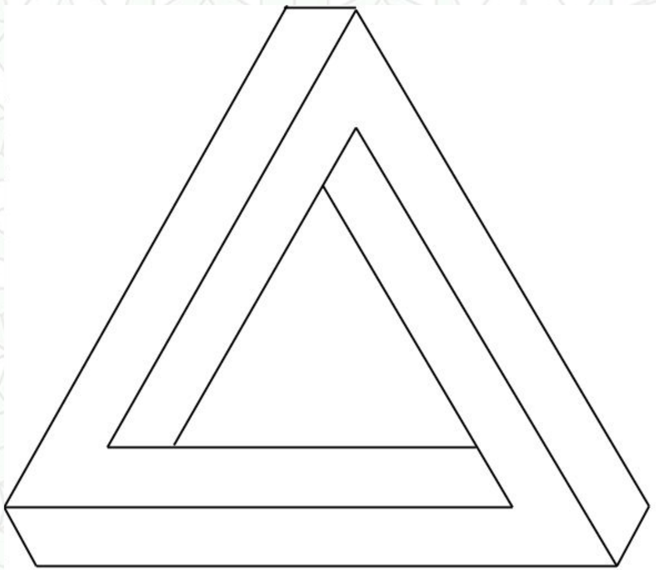


https://www.newworldencyclopedia.org/entry/necker_cube

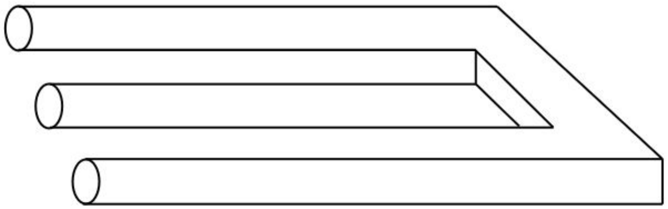
https://commons.wikimedia.org/wiki/File:Necker%27s_cube.svg

# Impossible Objects

- Penrose triangle

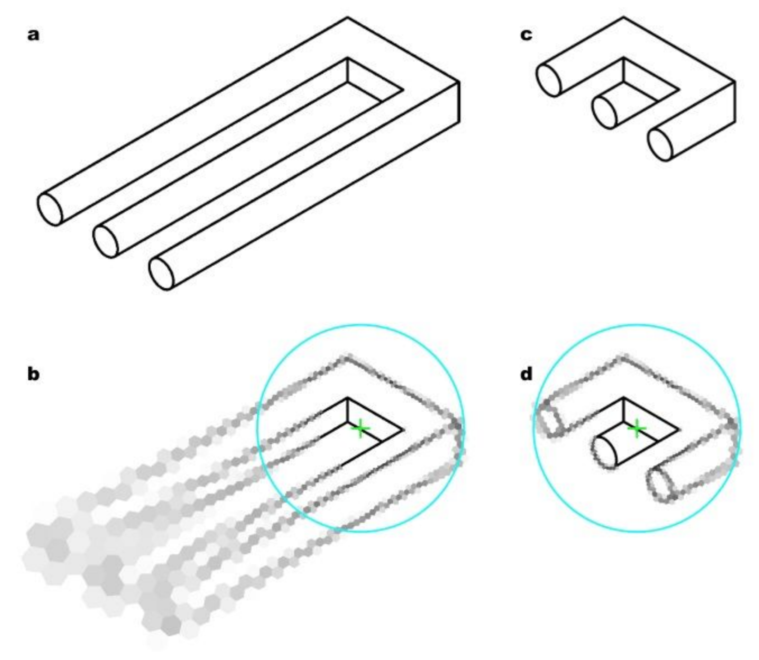


- Devil's tuning fork



https://simple.wikipedia.org/wiki/Impossible_object

# Impossible Objects

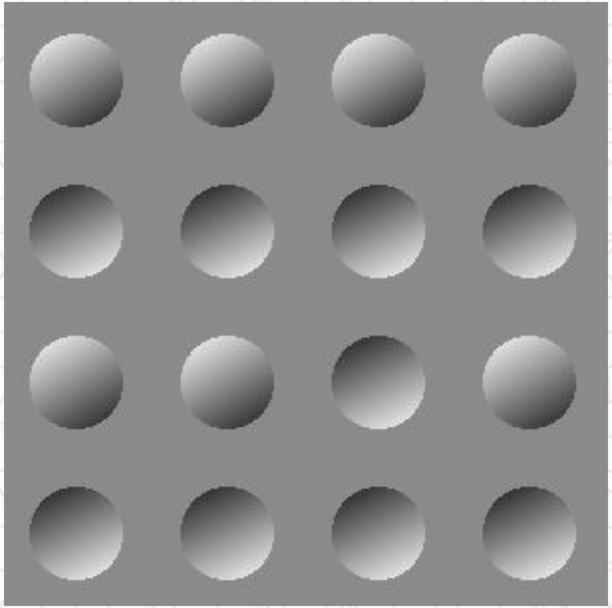



“Combining Deep Learning and Active Contours Opens The Way to Robust, Automated Analysis of Brain Cytoarchitectonics”, Thierbach et al, 2018

Belvedere
M.C. Escher
1958

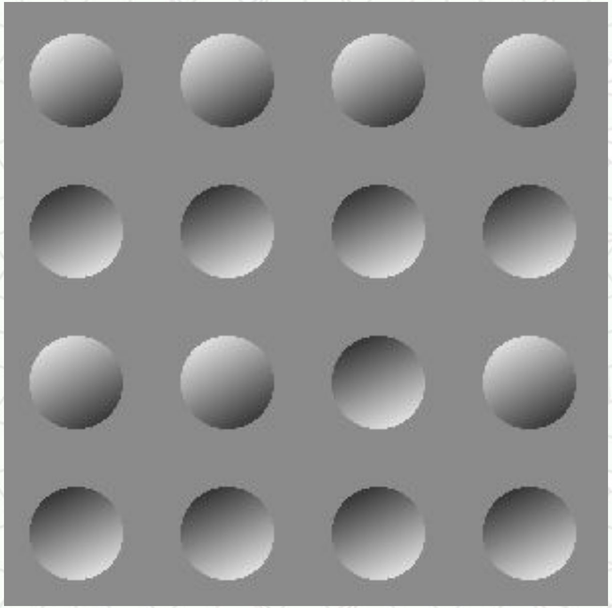
# Bump or Divot?

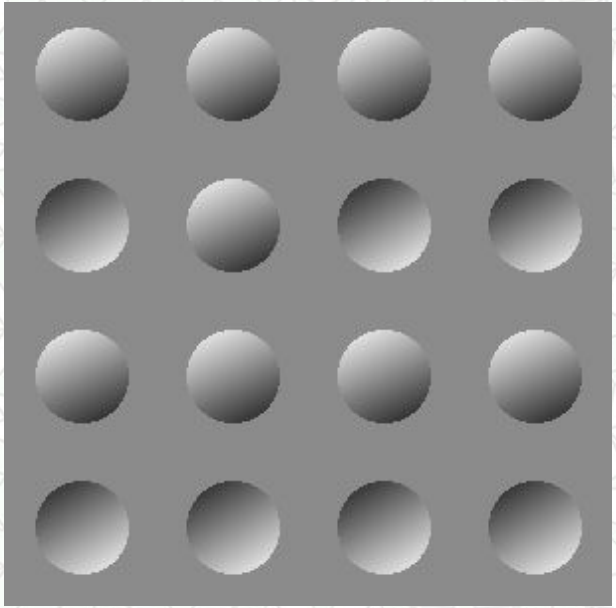
- How many dots are raised higher than the surrounding surface?
- How many are indented pushed in/lower than the surrounding surface?



# Bump or Divot?

- How many dots are raised higher than the surrounding surface?
- How many are indented pushed in/lower than the surrounding surface?

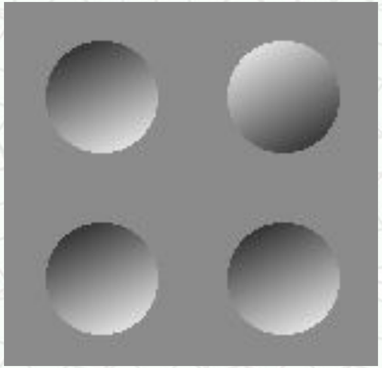


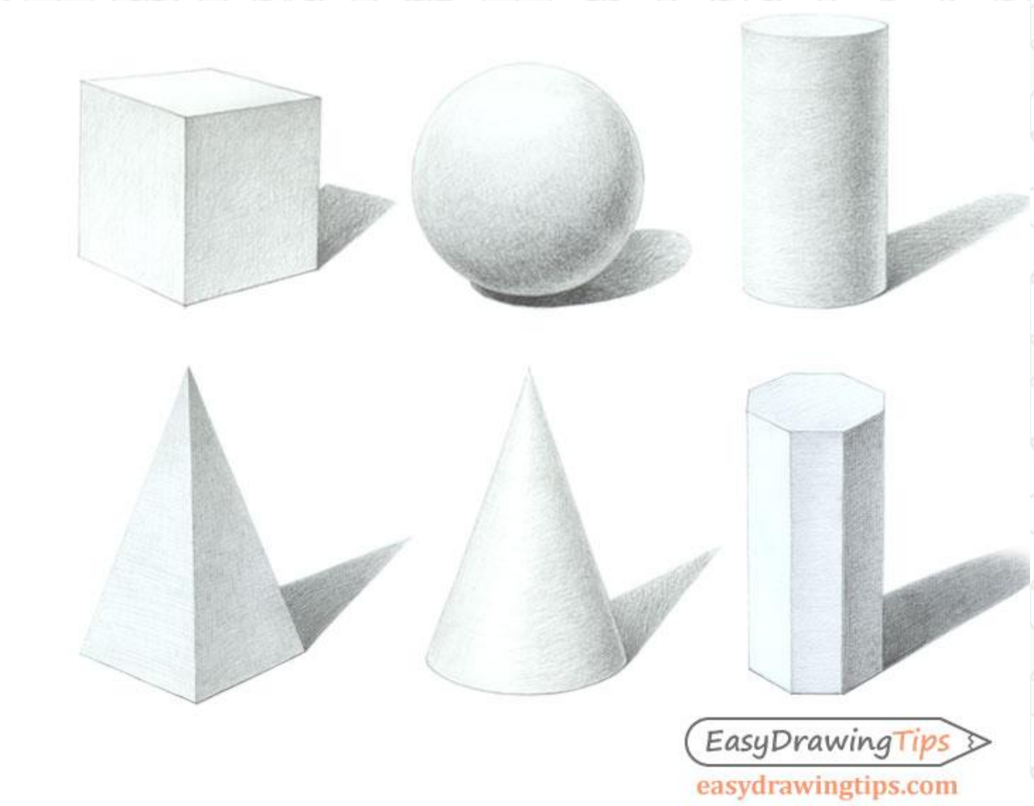


*this is the same image, just rotated 180°*

# Lighting Assumptions

- Where is the light source in 3D?
- Why do we assume that?

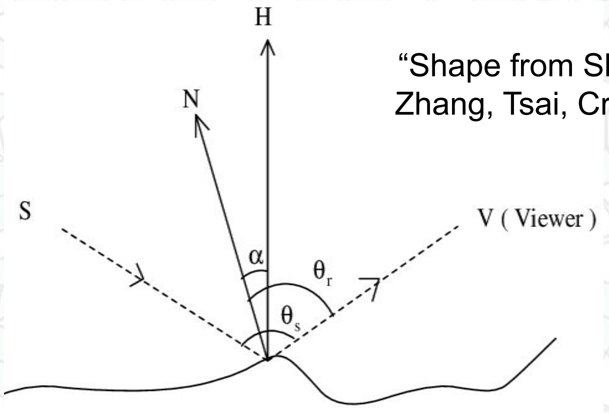




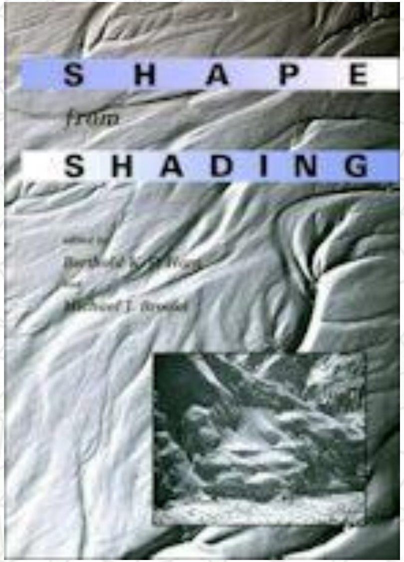

https://www.easydrawingtips.com/shading-basic-3d-shape-drawings-tutorial/

# Shape from Shading

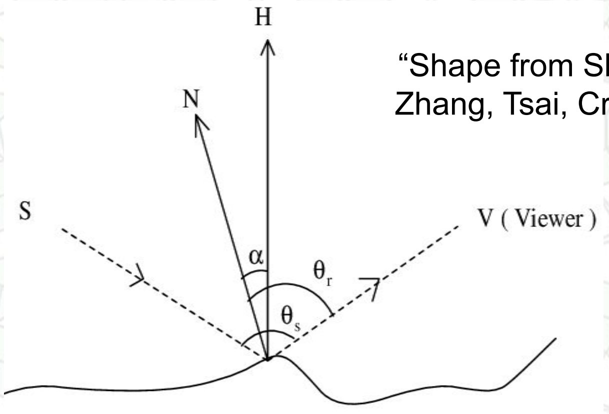
- Surface normal + light position → *greyscale pixel value*




“Shape from Shading: A Survey”
Zhang, Tsai, Cryer & Shah, 1999



# Shape from Shading

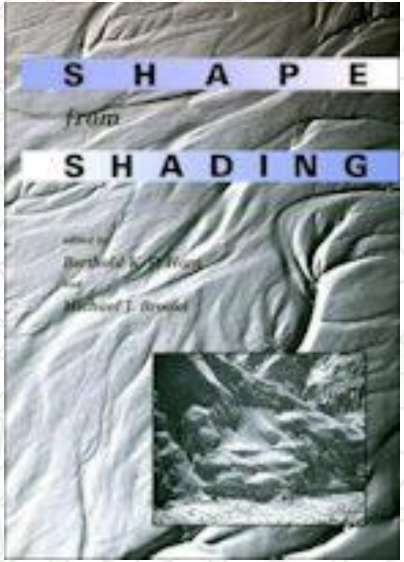



"Shape from Shading: A Survey"
Zhang, Tsai, Cryer & Shah, 1999

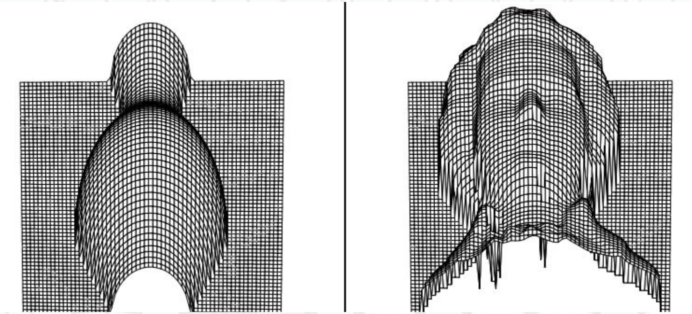
- Surface normal + light position → *greyscale pixel value*

- Assumption: surface is smooth (normal/gradient changes slowly)

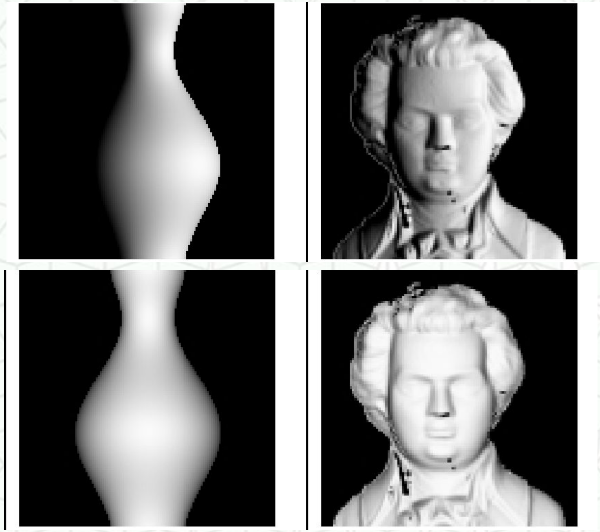
- Given a light position + pixel color → *a set of possible surface normals (not unique)*
- Given 2 light positions + 2 pixel colors → *constrained to one possible surface normal!*
- Reverse engineer a global smooth & connected surface that matches the estimated surface normal



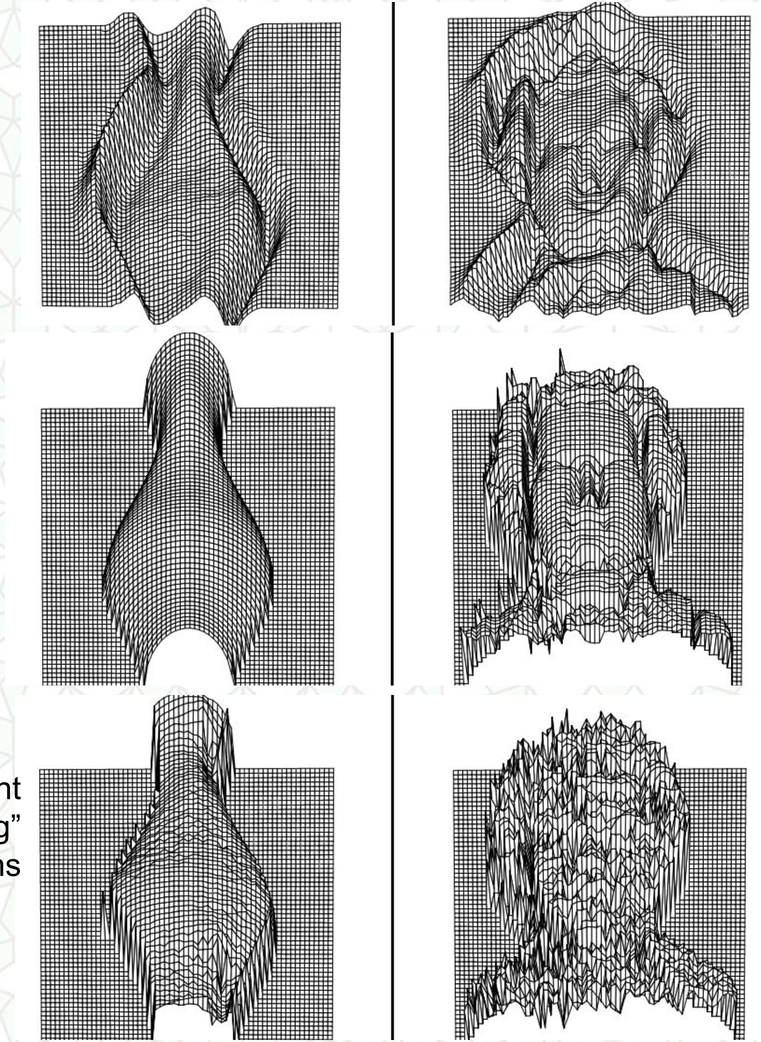
# Shape from Shading



synthetic
3D input
meshes



renderings of
input mesh
from 2 different
lighting positions



output from different
“shape from shading”
algorithms

“Shape from Shading: A Survey”
Zhang, Tsai, Cryer & Shah, 1999

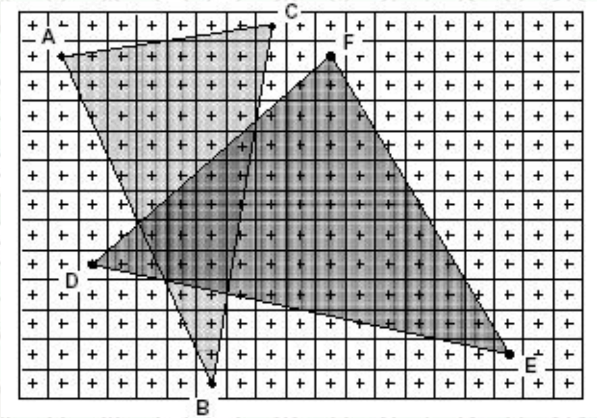
# Outline for Today

- Homework 7 & Final Project Proposal Feedback
- Last Time: General Position, Robustness, & Exact Computation
- Line Drawings & Early Computer Vision / AI
- Hidden Line Drawing: z-Buffer
- Hidden Line Drawing: Painter's Algorithm
- Binary Space Partition
- Binary Space Partition Analysis
- Discussion & Comparison to Quad Tree & kD Tree
- Final Project 3D Visualization ChallengesNext Time: Polyominoes & Tiling

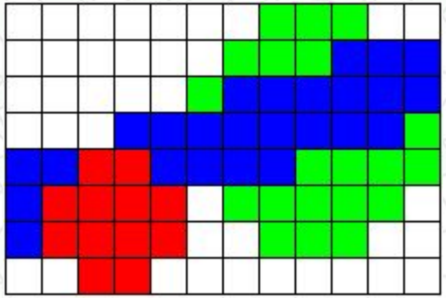
# Hidden Line Drawing / Depth Buffer (z-Buffer)

- Given a primitive's vertices & the color / illumination at each vertex:
- Figure out which pixels to "turn on" to render the primitive
- Interpolate the color / illumination values to "fill in" the primitive
- At each pixel, keep track of the closest primitive (depth buffer / z-buffer)

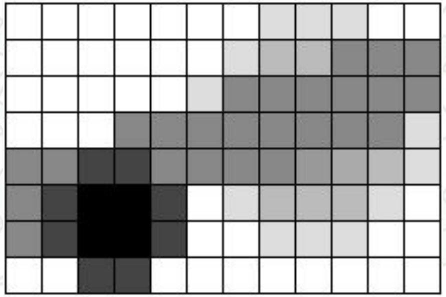
***Triangles can be in any order!***
***A.k.a. "Polygon soup"***




```
glBegin(GL_TRIANGLES)
glNormal3f(...)
glVertex3f(...)
glVertex3f(...)
glVertex3f(...)
glEnd();
```



**frame buffer**



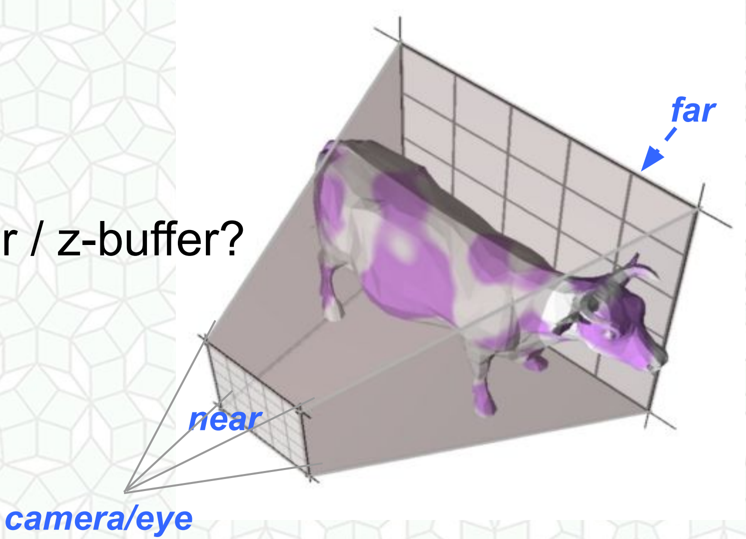
**depth buffer**

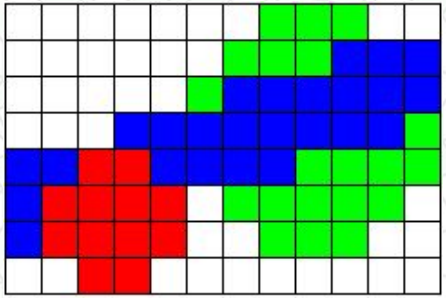
# Scan Conversion / Rendering Pipeline

- Running time of depth buffer / z-buffer?

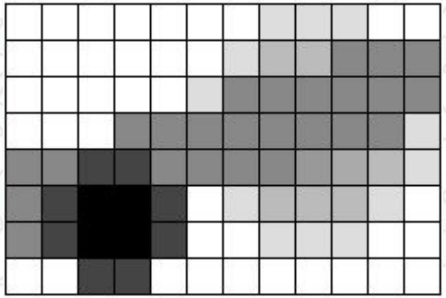
- Extra memory use for depth buffer / z-buffer?

- Flaws with depth buffer / z-buffer?



frame buffer



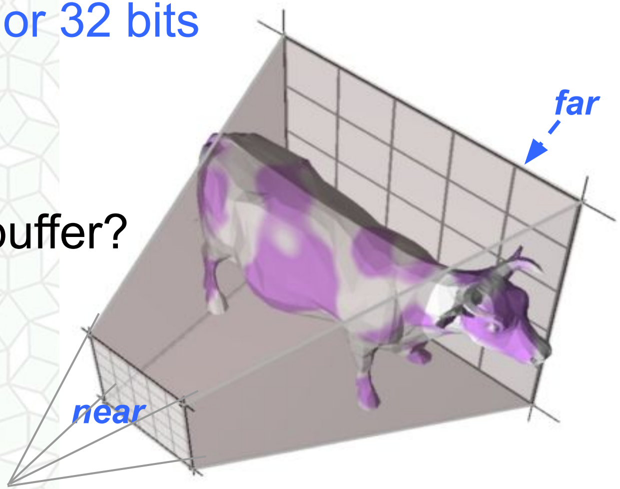
depth buffer

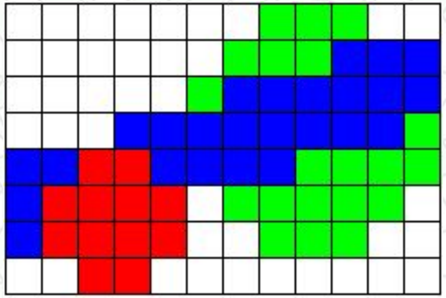
# Scan Conversion / Rendering Pipeline

- Running time of depth buffer / z-buffer?
  → *O(n * w * h)* worst case large triangles
  → *O(n)* in practice

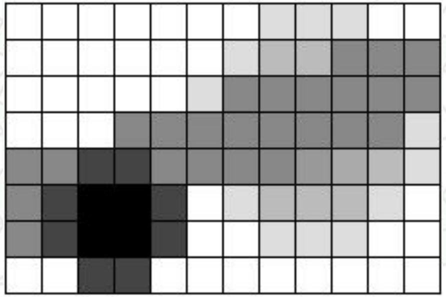
- Extra memory use for depth buffer / z-buffer?
  → *O(w*h)* * 8 bits or 24 bits or 32 bits
  *In early graphics, this was too expensive to consider!*

- Flaws with depth buffer / z-buffer?
  - Limited precision
  - Need to choose near & far plane carefully



**frame buffer**



**depth buffer**

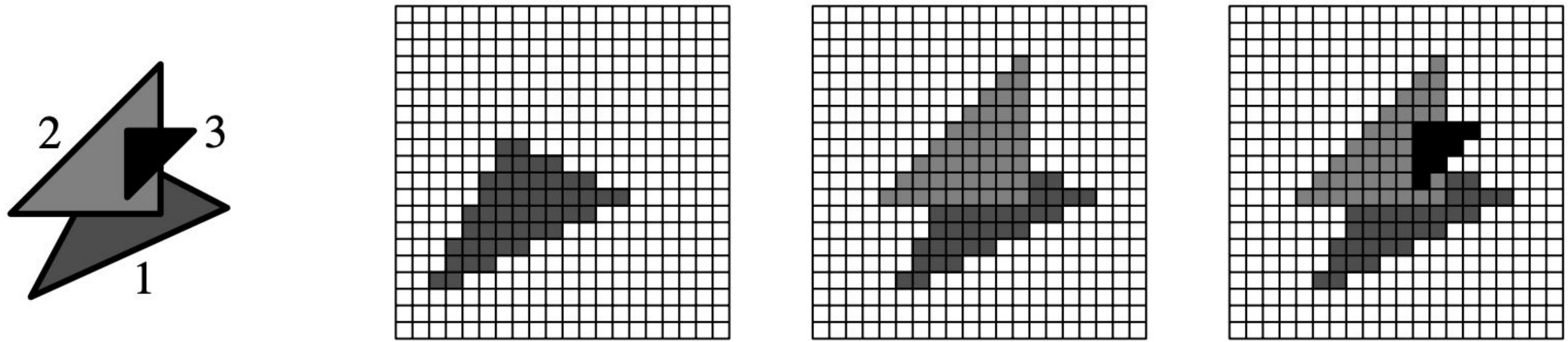
# Outline for Today

- Homework 7 & Final Project Proposal Feedback
- Last Time: General Position, Robustness, & Exact Computation
- Line Drawings & Early Computer Vision / AI
- Hidden Line Drawing: z-Buffer
- Hidden Line Drawing: Painter's Algorithm
- Binary Space Partition
- Binary Space Partition Analysis
- Discussion & Comparison to Quad Tree & kD Tree
- Final Project 3D Visualization Challenges
- Next Time: Polyominoes & Tiling

# Hidden Line Drawing: Painter's Algorithm

- Let's order the primitives by how close they are to the camera
- Draw the primitives from back to front
- Then we don't need to keep track of the depth!

  *Save memory!*



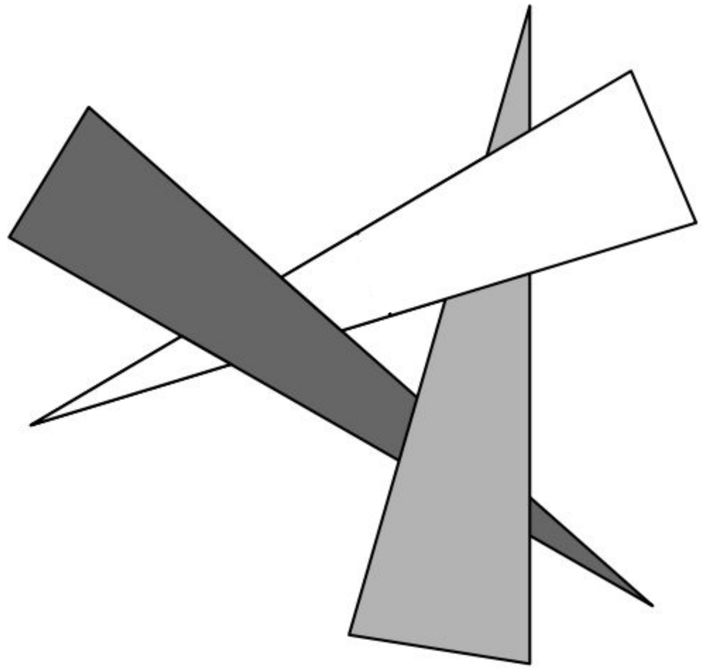

*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 12

**Bob Ross -**
**Peaceful Waters**
**Season 3**
**Episode 13**

**https://www.twoinchbrush.com/painting/peaceful-waters**

# Hidden Line Drawing: Painter's Algorithm

- Let's order the primitives by how close they are to the camera
- Draw the primitives from back to front

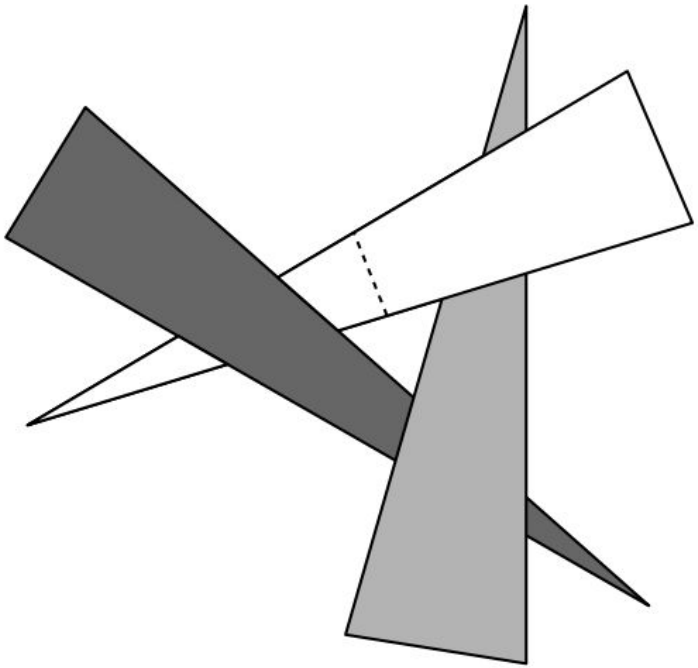


*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 12

# Hidden Line Drawing: Painter's Algorithm

- Let's order the primitives by how close they are to the camera
- Draw the primitives from back to front

- Warning: Object layering may be complex and have cycles
  E.g., a > b, b > c, c > a
- *Solution: Split primitives as necessary to break cycles*



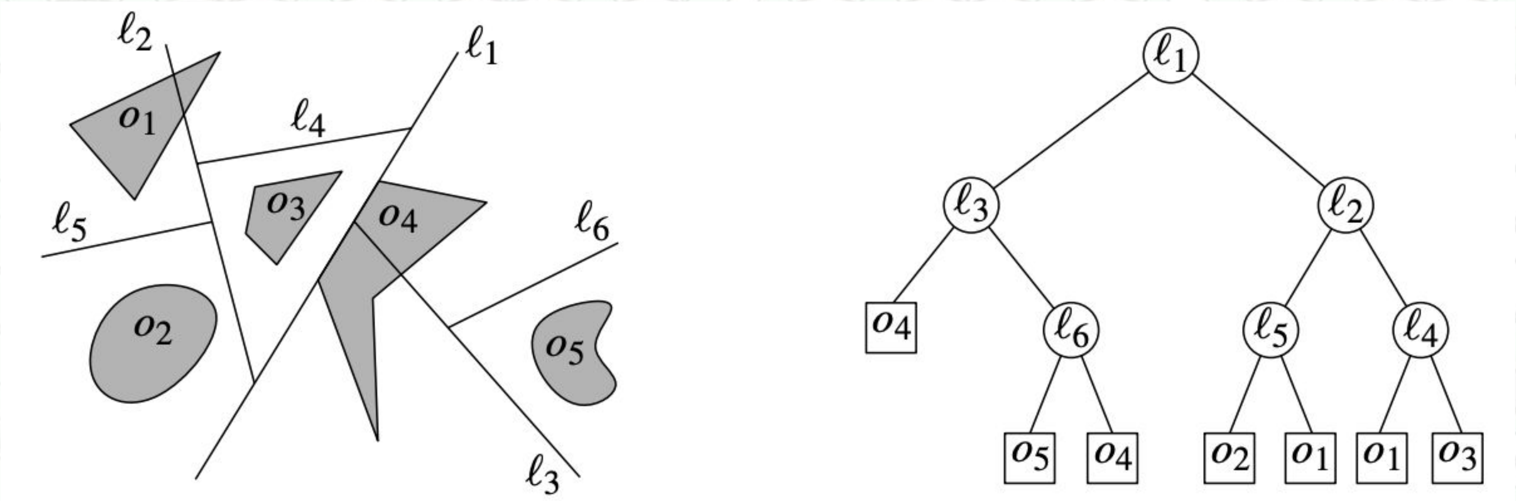
*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 12

# Outline for Today

- Homework 7 & Final Project Proposal Feedback
- Last Time: General Position, Robustness, & Exact Computation
- Line Drawings & Early Computer Vision / AI
- Hidden Line Drawing: z-Buffer
- Hidden Line Drawing: Painter's Algorithm
- Binary Space Partition
- Binary Space Partition Analysis
- Discussion & Comparison to Quad Tree & kD Tree
- Final Project 3D Visualization Challenges
- Next Time: Polyominoes & Tiling

# Definition: Binary Space Partition

- Place items in a binary tree, each node stores a half plane
- Primitives that are collinear with the half plane are stored in the node
- Items overlapping a half plane are copied/split into two primitives
- We recurse until exactly one item is left, it is stored in the leaf

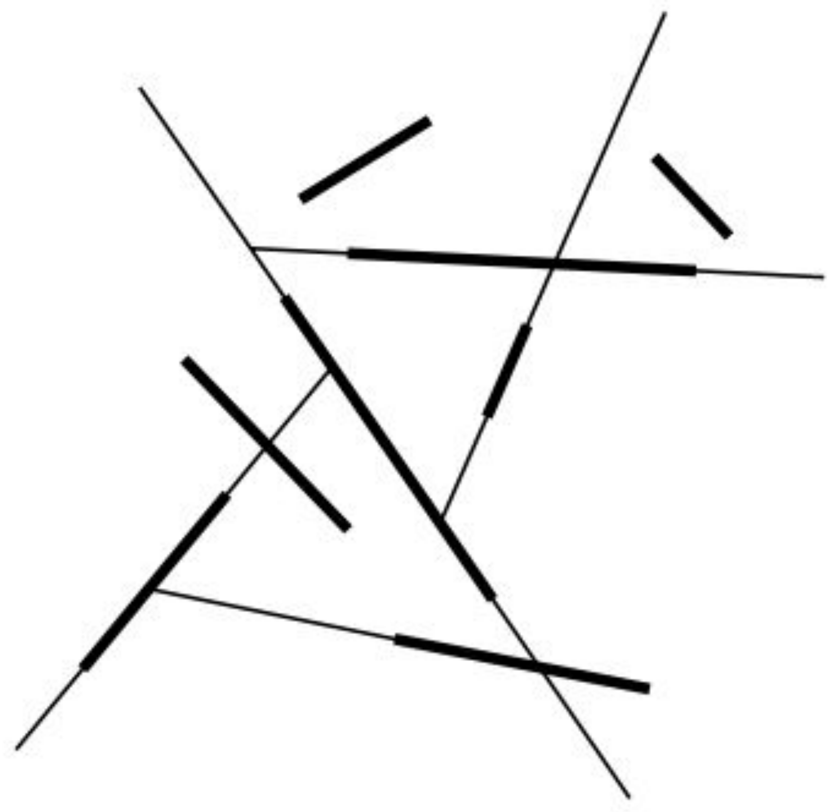



*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 12

# Auto-Partition

- In practice, it is common to use the primitives as half-planes
- If a BSP only uses half-planes derived from the input data, it is called an auto-partition

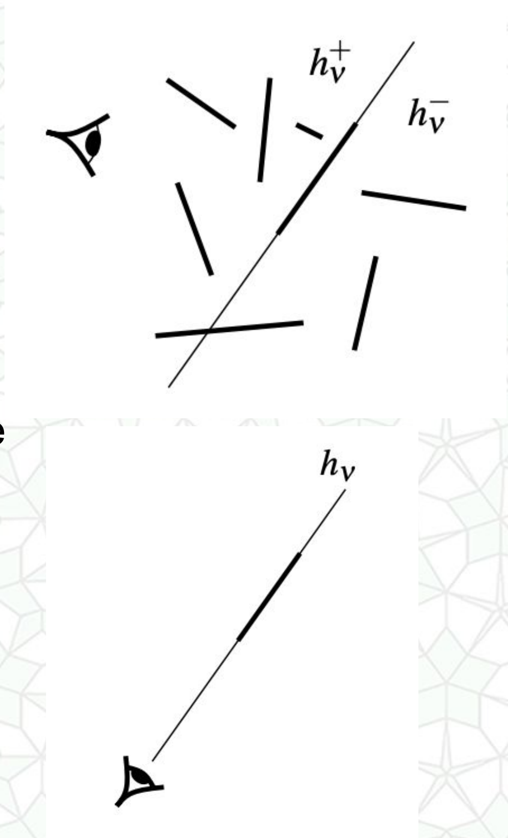
- Primitive is stored at the node (rather than pushed down to a leaf)
  - So it will probably be smaller…
  - But the optimal partitioning (minimal # of nodes) may require hyperplanes that are not derived from the input!



*Computational Geometry Algorithms and Applications*, de Berg, Cheong, van Kreveld and Overmars, Chapter 12

# Using a BSP to Render via Painter’s Algorithm

- If we’re at a leaf,
  - Render items in current node
- Else if camera to **left** of current node hyperplane
  - Recurse to right of current node
  - Render items in current node
  - Recurse to left of current node
- Else if camera is to **right** of current node hyperplane
  - Recurse to left of current node
  - Render items in current node
  - Recurse to right of current node
- Else we’re on the split plane
  (we can ignore items in current node)
  - Recurse to left of current node
  - Recurse to right of current node




*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 12

# Outline for Today

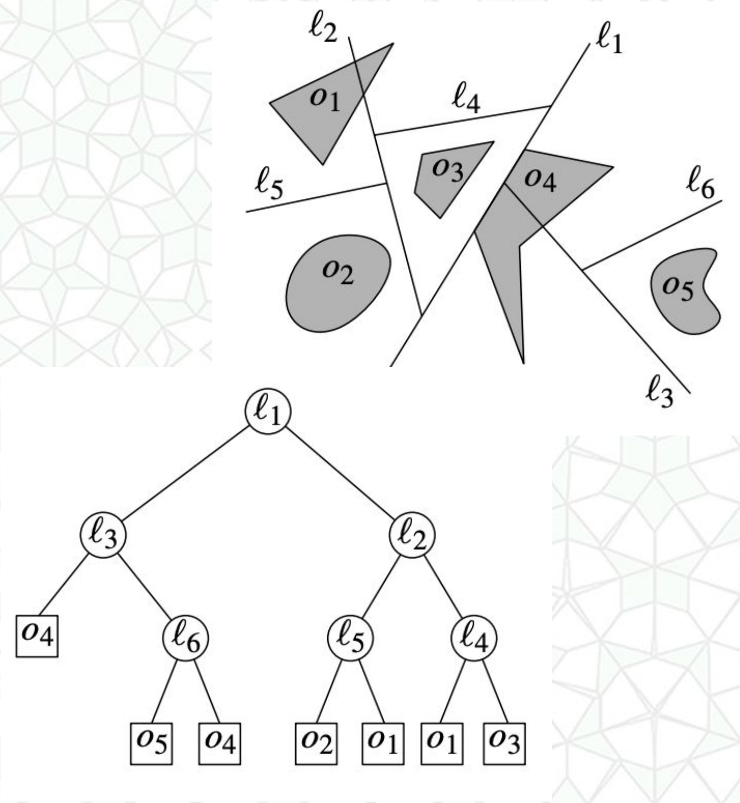
- Homework 7 & Final Project Proposal Feedback
- Last Time: General Position, Robustness, & Exact Computation
- Line Drawings & Early Computer Vision / AI
- Hidden Line Drawing: z-Buffer
- Hidden Line Drawing: Painter's Algorithm
- Binary Space Partition
- Binary Space Partition Analysis
- Discussion & Comparison to Quad Tree & kD Tree
- Final Project 3D Visualization Challenges
- Next Time: Polyominoes & Tiling

# Analysis: Using BSP for Painter's Algorithm

- For *n* non-intersecting primitives
- Best case:

- Worst case:

- Overall: Painter's algorithm

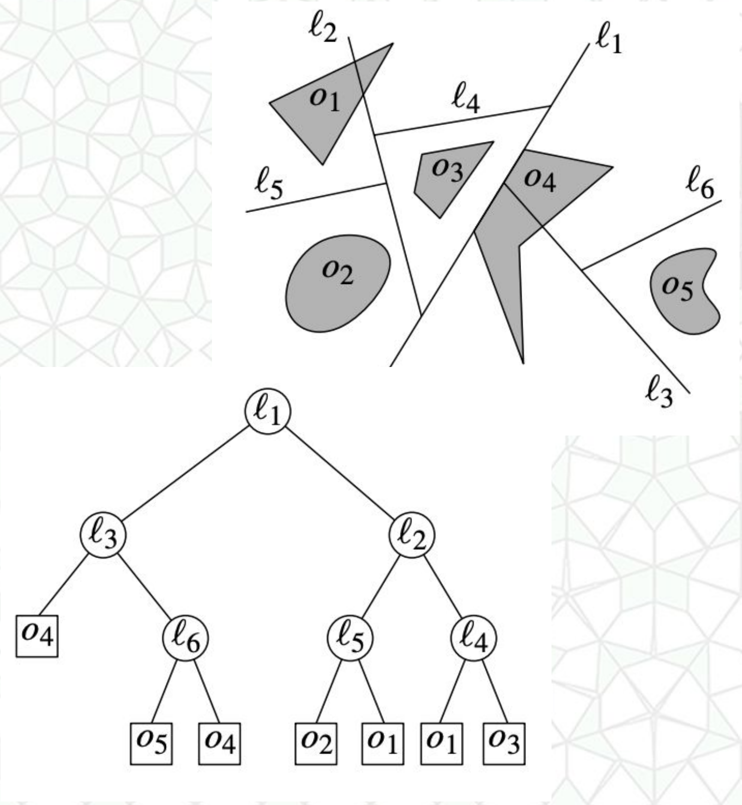



*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 12

# Analysis: Using BSP for Painter's Algorithm

- For *n* non-intersecting primitives
- Best case:
  - No primitives are split
  - O(*n*) nodes in the tree
  - Tree is perfectly balanced, height = *O(log n)*
- Worst case:
  - Every primitive is split by every plane
  - $O(n^2)$ nodes in the tree
  - Tree is unbalanced, height = *O(n)*
- Overall: Painter's algorithm
  - O(# of nodes in the tree)
  - (height is irrelevant!)

- ***Can we do better than worst case??***

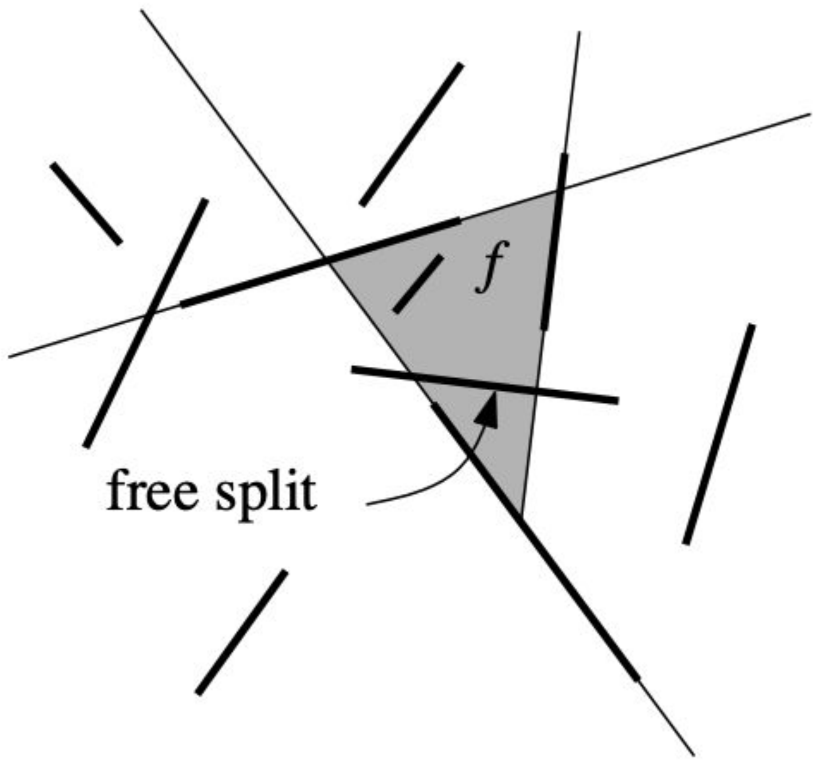



*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 12

# Small Optimization: “Free Split”

- Assumption: Our input primitives do not intersect

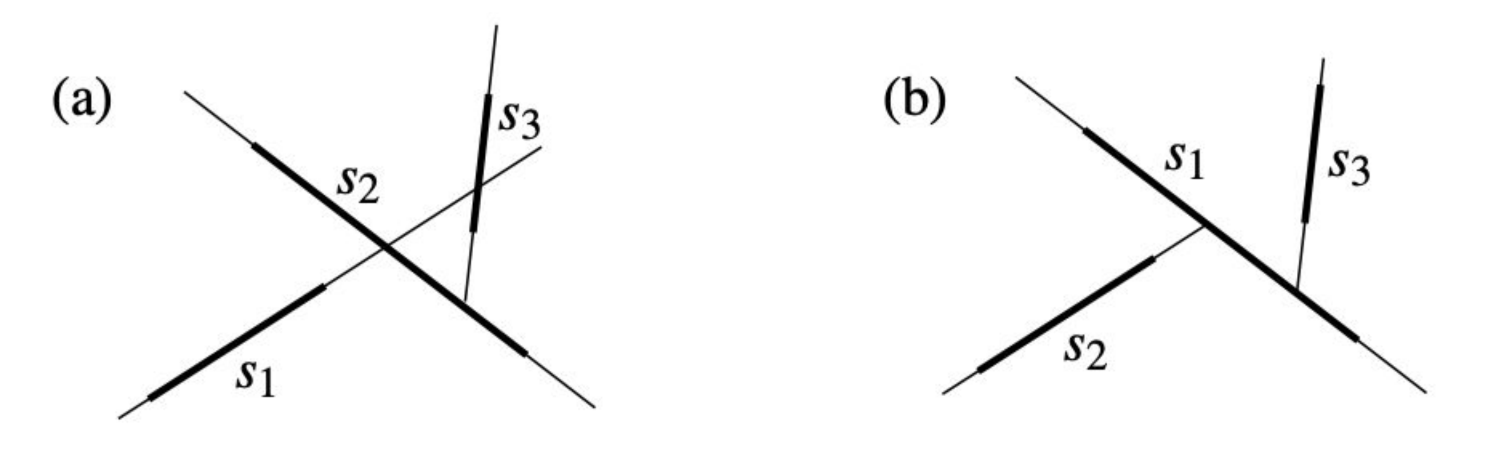
- If we can determine that both primitive endpoints are on the half plane boundaries of the current subtree
- Choosing that primitive as the next half plane node is guaranteed not to split any primitives




*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 12

# Randomized Incremental Construction

- Note: Some orderings are better than others: (result in fewer split primitives)




- *Let's randomize the order!*

*Computational Geometry Algorithms and Applications*, de Berg, Cheong, van Kreveld and Overmars, Chapter 12

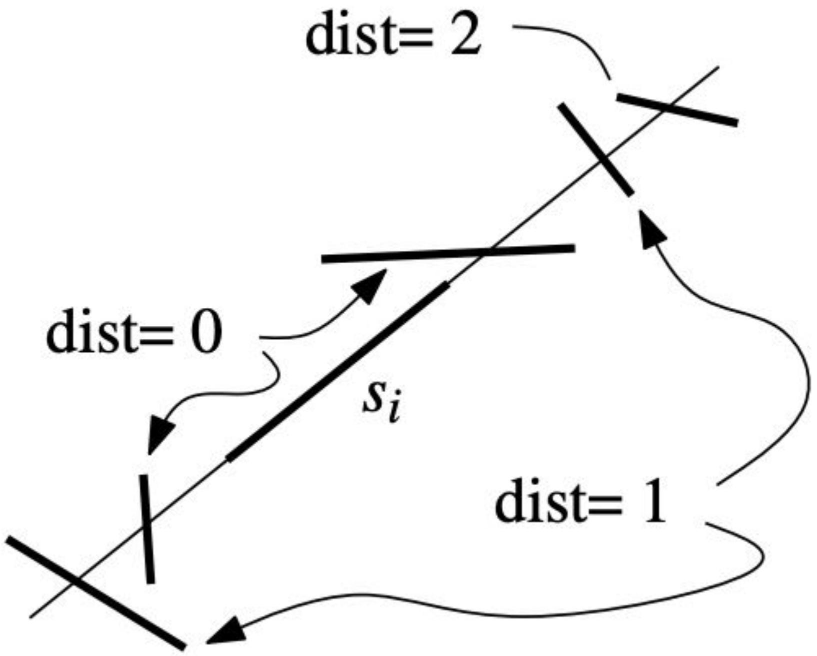
# Randomized Incremental Construction

- Let's randomize the order!

  $s_0, s_1, s_2, \ldots . \; s_i \ldots . \; s_k \ldots$

- What's the chance that a primitive $s_k$ will be split by the half plane derived from $s_i$?

- If there are many other segments between $s_i$ and $s_k$ there is a good chance one of them will shield $s_k$ from being split by $s_i$




*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 12

# Randomized Incremental Construction

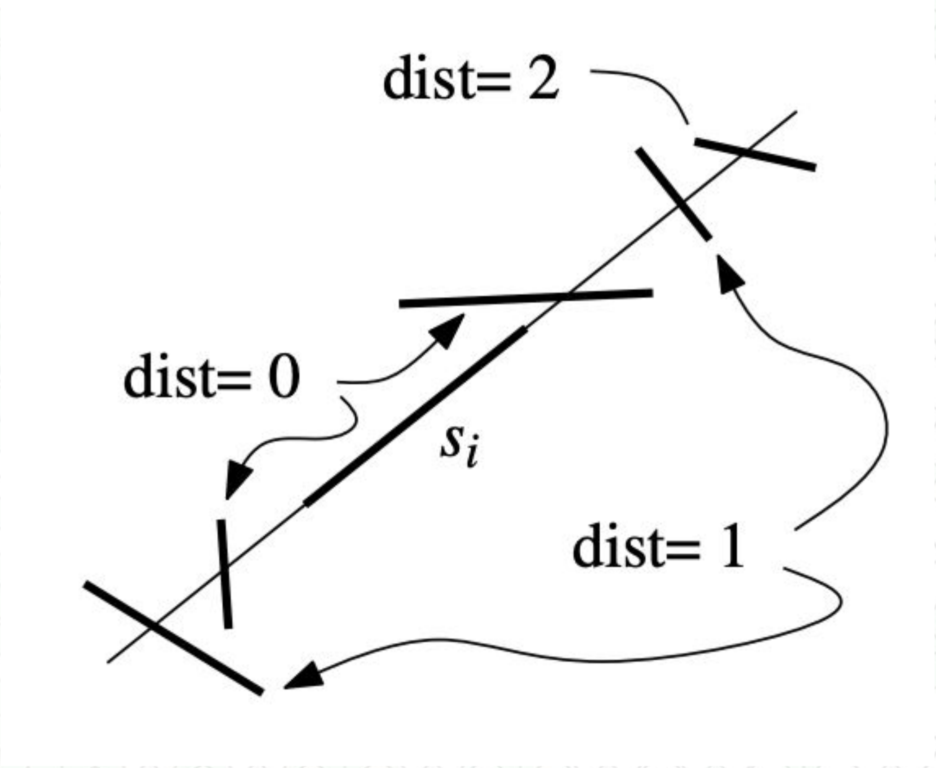
- Let's randomize the order!

  $s_0, s_1, s_2, \ldots. s_i \ldots. s_k \ldots$

- Randomized BSP
  can be shown to be
  have *O(n log n)* nodes

- And can be constructed
  in $O(n^2 \log n)$

- *Which is better than our worst case*
  *But still doesn't seem great…*




*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 12

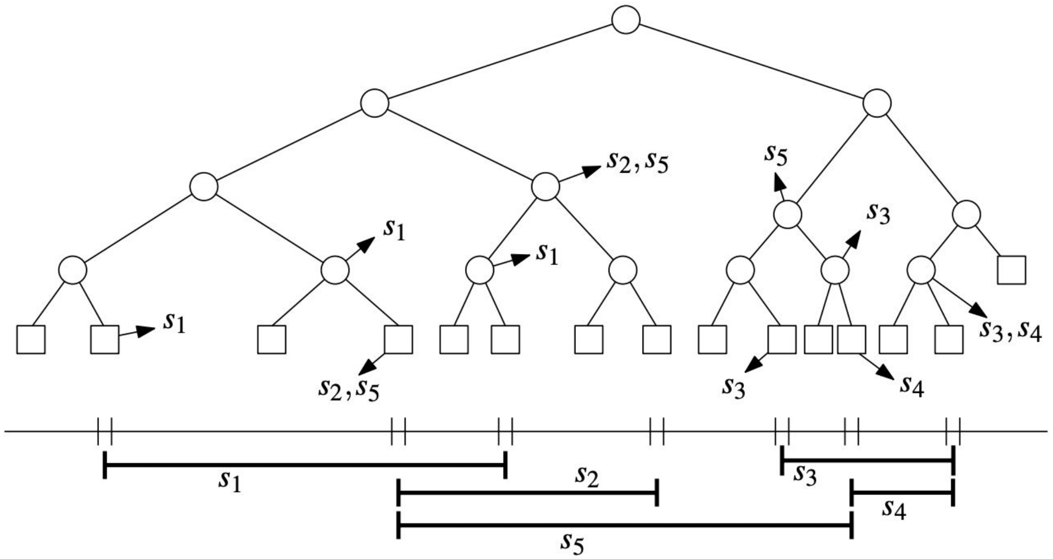
# Review of Segment Tree - (Lecture 16)

- For *n* input segments, for a query that will return *k* segments

- Memory:
  Each segment is stored in
  at most 2 nodes per level
  → *O(n log n)*

- Construction Time:
  Presort all endpoints
  by x & y *O(n log n)*
  → *O(n log n)*

- Query Time:
  → *O(log n * log n + k)*
  → *O*($\log^2 n + k$)

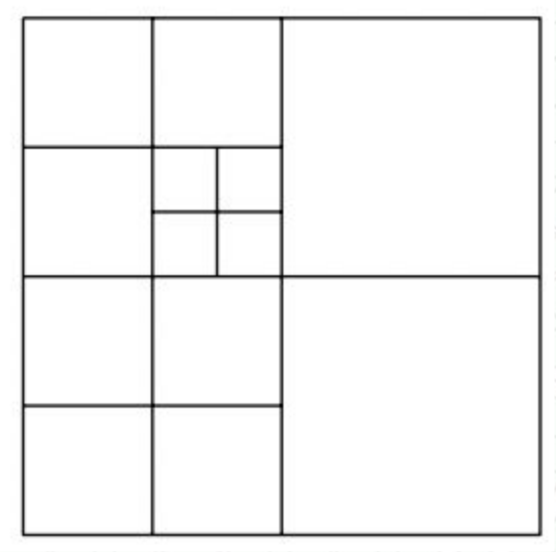



*Computational Geometry Algorithms and Applications*,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

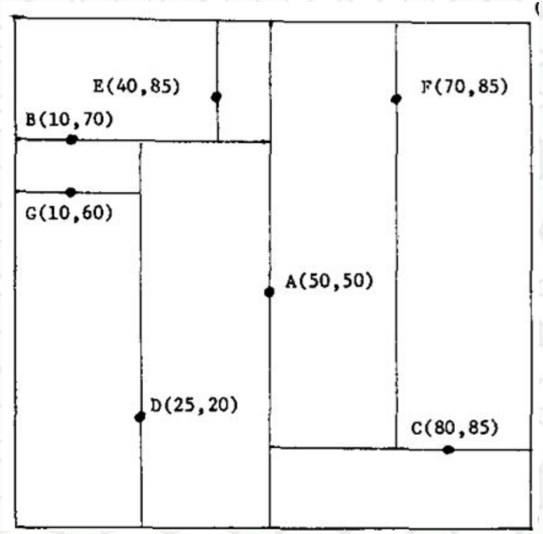
# Outline for Today

- Homework 7 & Final Project Proposal Feedback
- Last Time: General Position, Robustness, & Exact Computation
- Line Drawings & Early Computer Vision / AI
- Hidden Line Drawing: z-Buffer
- Hidden Line Drawing: Painter's Algorithm
- Binary Space Partition
- Binary Space Partition Analysis
- Discussion & Comparison to Quad Tree & kD Tree
- Final Project 3D Visualization Challenges
- Next Time: Polyominoes & Tiling

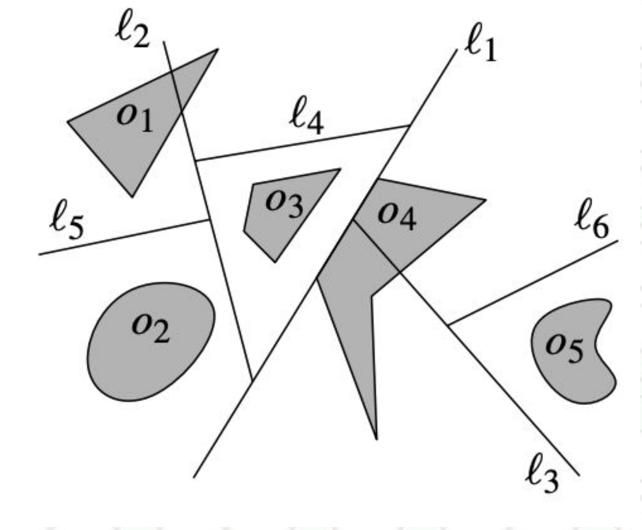
# Discussion - Quad Tree, kD Tree, BSP



**Quad Tree**



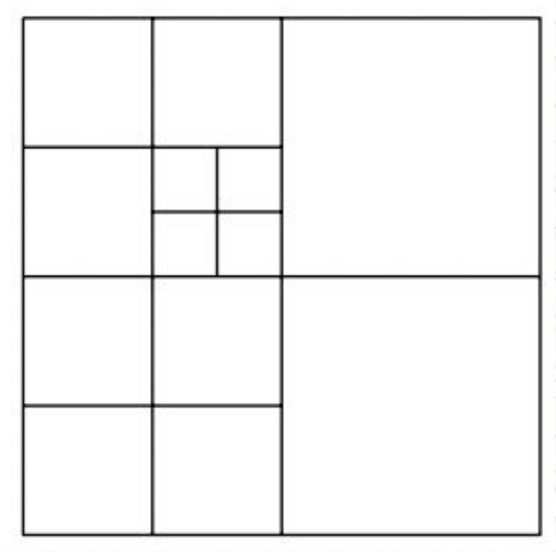

**k-D Tree**



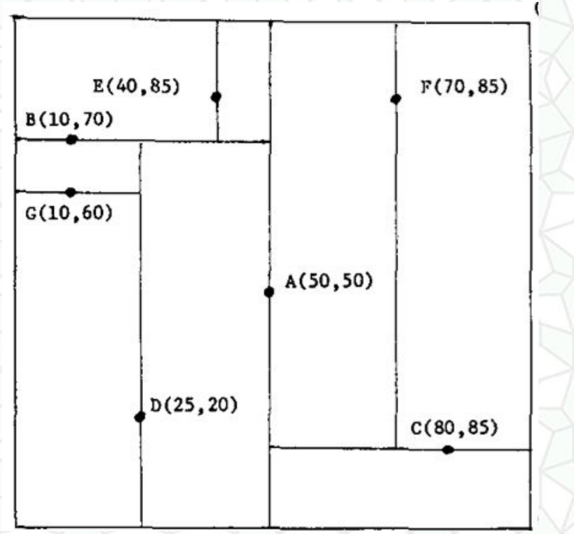

**BSP**

# Discussion - Quad Tree, kD Tree, BSP

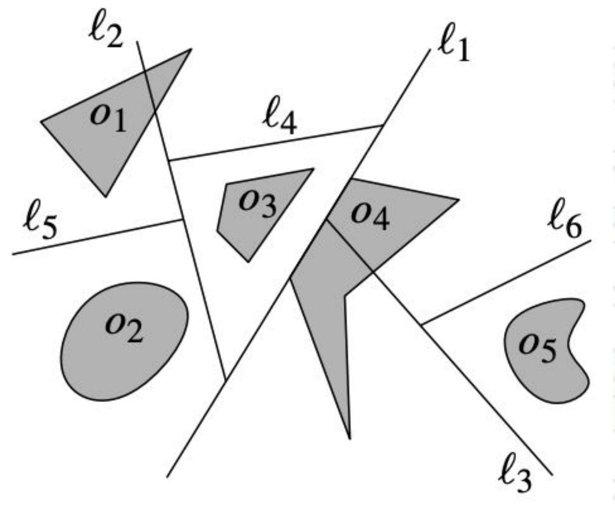
- k-D trees are a special case of BSP (where splits are always axis aligned)
- Quad trees are a special case of k-D trees
  (where splits are always at the midpoints)



**Quad Tree**



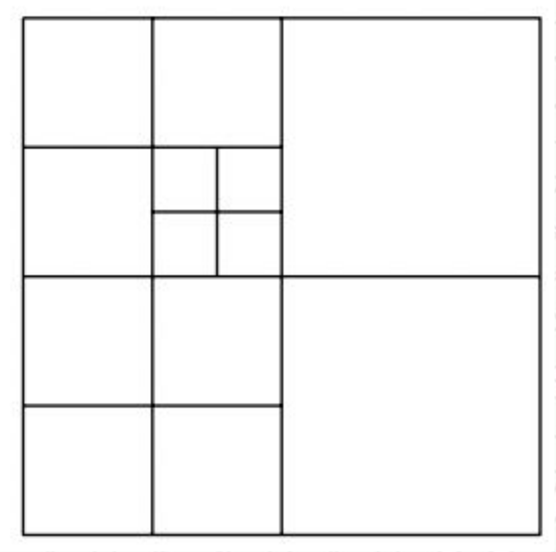

**k-D Tree**



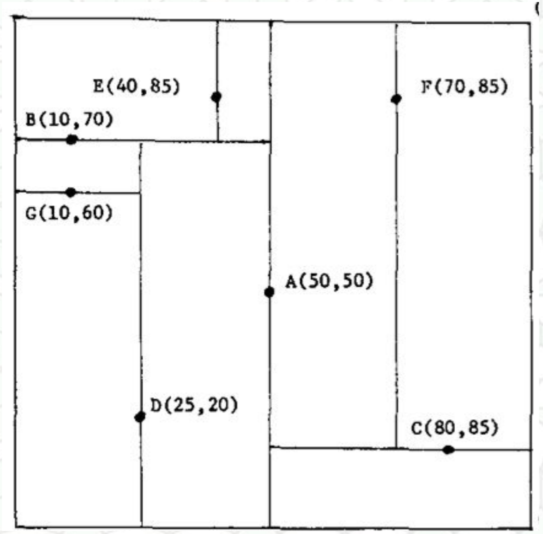

**BSP**

# Discussion - Quad Tree, kD Tree, BSP

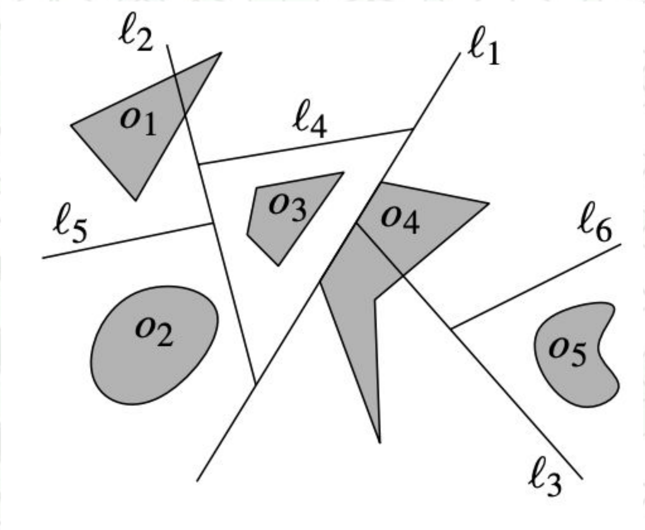
- Points (zero area) can be stored efficiently in any of these structures
- Items that have dimension and overlap split point are more complicated



Quad Tree



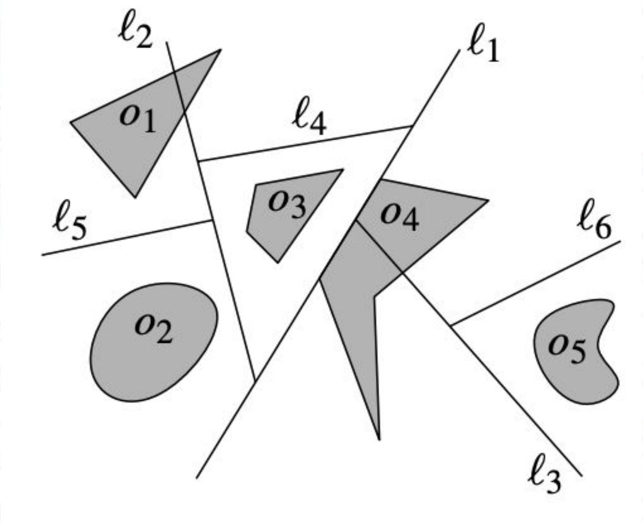

k-D Tree




BSP

# Discussion - BSP & Low Density Scenes

- BSP are harder to visualize, and therefore perhaps harder to intuitively understand, debug, and analyze
- Usually the performance of a BSP is much better than the conclusion reached by randomized analysis.
- Why?
  - In practice most objects are relatively small
  - In practice density of objects in a scene is sparse
  - Therefore it is likely the objects can be separated by planes without requiring the expected worst case number of splits
- *For more details, see analysis in the book…*

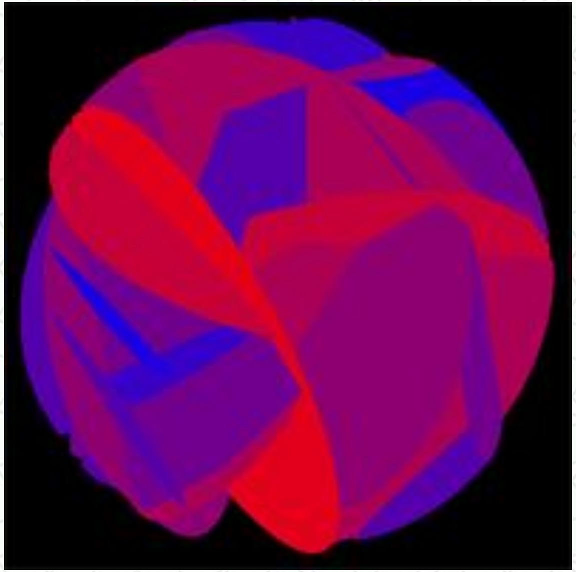



*Computational Geometry Algorithms and Applications*, de Berg, Cheong, van Kreveld and Overmars, Chapter 12

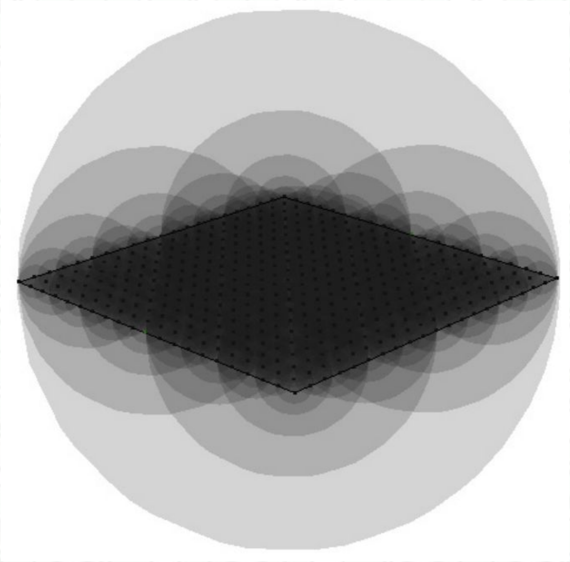
# Outline for Today

- Homework 7 & Final Project Proposal Feedback
- Last Time: General Position, Robustness, & Exact Computation
- Line Drawings & Early Computer Vision / AI
- Hidden Line Drawing: z-Buffer
- Hidden Line Drawing: Painter's Algorithm
- Binary Space Partition
- Binary Space Partition Analysis
- Discussion & Comparison to Quad Tree & kD Tree
- Final Project 3D Visualization Challenges
- Next Time: Polyominoes & Tiling

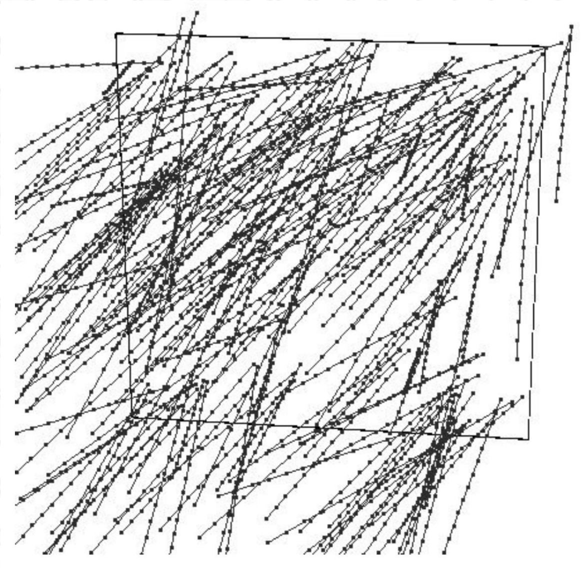
# Static Visualization of 3D Structures is Challenging!



Visualization of Binary Space Partition
Casey Shields, 2007



Bounding Spheres for Collision Detection
Fangyuan Ding, 2013



Hair Simulation
Helen Lei Zefanya Putri, 2017

# Final Project (Visualization/User Interface) Suggestions

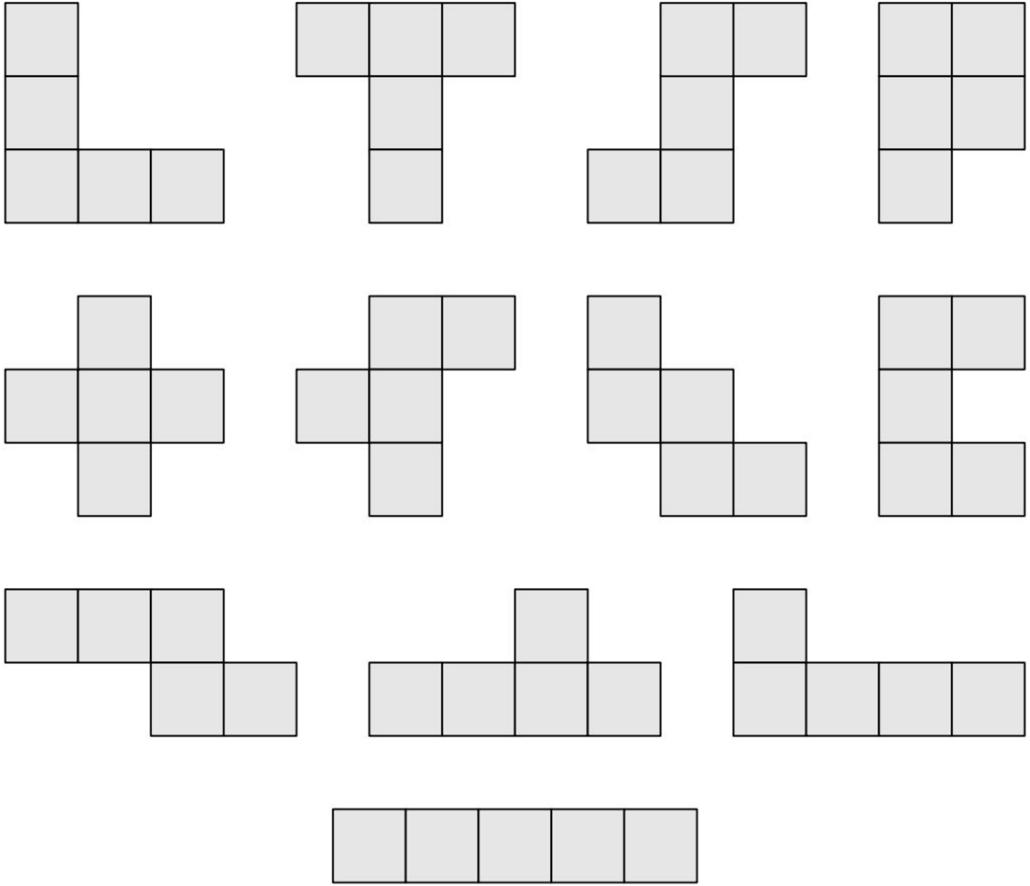
- 3D is difficult
  - Recommended to start with very simple examples in 2D
  - Ok to limit yourself to 2D (it's a short project)
- Building high quality, intuitive user interfaces is challenging and really time consuming
  - Recommended to skip building a fancy user interface
- Visualization / diagramming is important for debugging
- Visualization / diagramming is important for communication
  - How will you communicate your project to your peers? (Our last day of class is Final Project Presentations!)
  - Will you give a live demo of your project during your presentation?
  - What images / screenshots / diagrams / graphs of data will you include in your final project report?

# Outline for Today

- Homework 7 & Final Project Proposal Feedback
- Last Time: General Position, Robustness, & Exact Computation
- Line Drawings & Early Computer Vision / AI
- Hidden Line Drawing: z-Buffer
- Hidden Line Drawing: Painter's Algorithm
- Binary Space Partition
- Binary Space Partition Analysis
- Discussion & Comparison to Quad Tree & kD Tree
- Final Project 3D Visualization Challenges
- Next Time: Polyominoes & Tiling

# Next Time: Polyominos

- There are 12 unique 5-ominoes (a.k.a. pentominoes)



“Ch 14: Polyominoes”, Barequet, Golomb, & Klarner, Handbook of Discrete and Computational Geometry, 2018