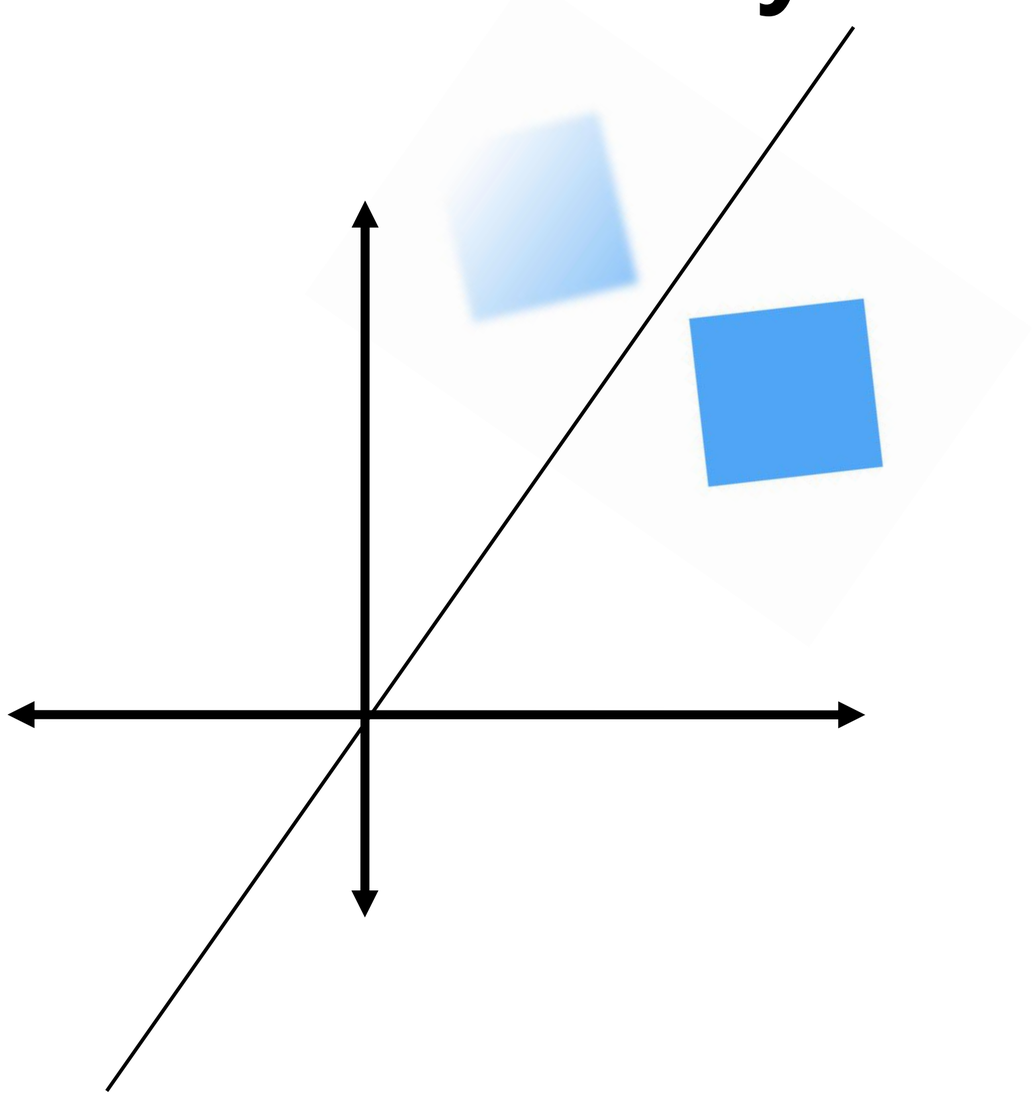
Perspective Projection Transformations, Geometry and Texture Mapping

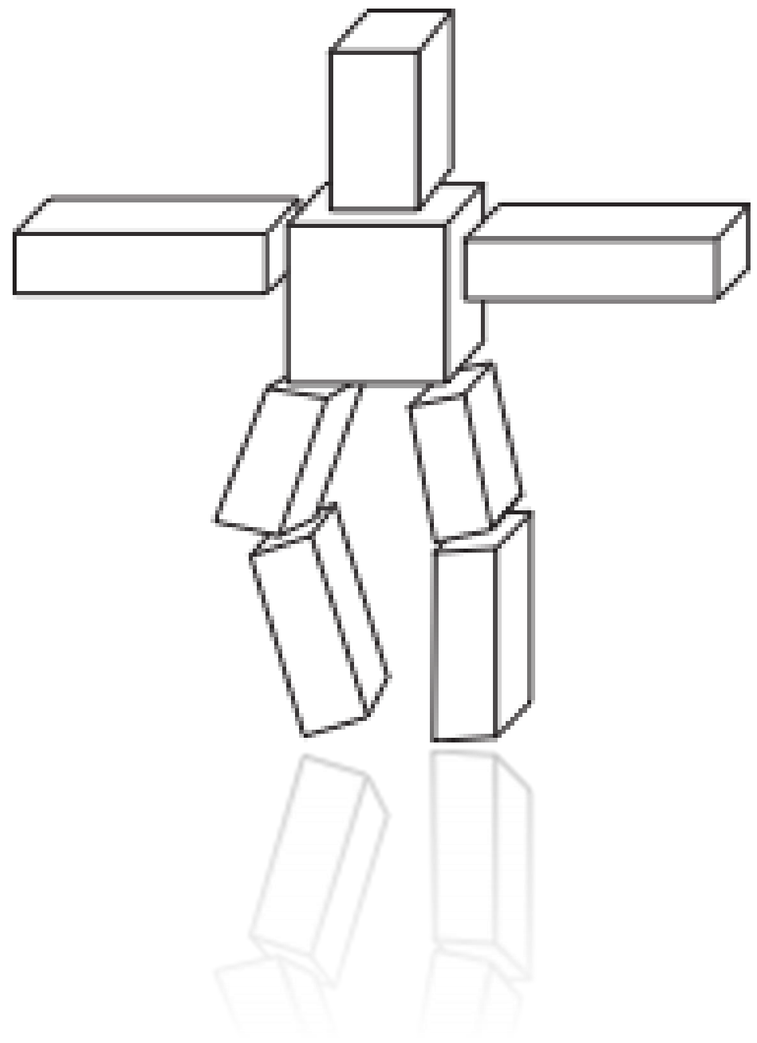
Exercise

- **Reflection about an arbitrary line**



Summary from last class

- **Transformations can be interpreted as operations that move points in space**
  - **e.g., for modeling, animation**



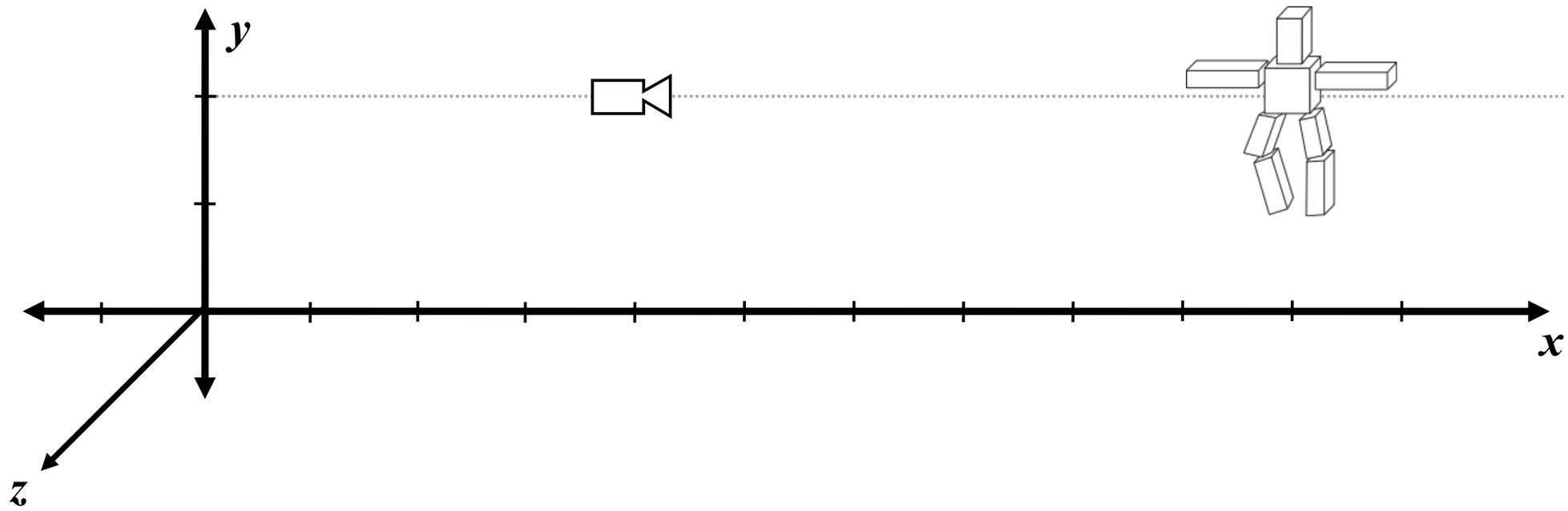
- **Or as a change of coordinate system**

- **Construct complex transformations as compositions of basic transforms**

- **Homogeneous coordinates allow non-linear transforms (e.g., affine, perspective projection) to be expressed as matrix-vector operations (linear transforms)**
  - **Matrix representation affords simple implementation and efficient composition**

Simple camera transform

- Consider object in world at (10, 2, 0)
- Consider camera at (4, 2, 0), looking down x axis



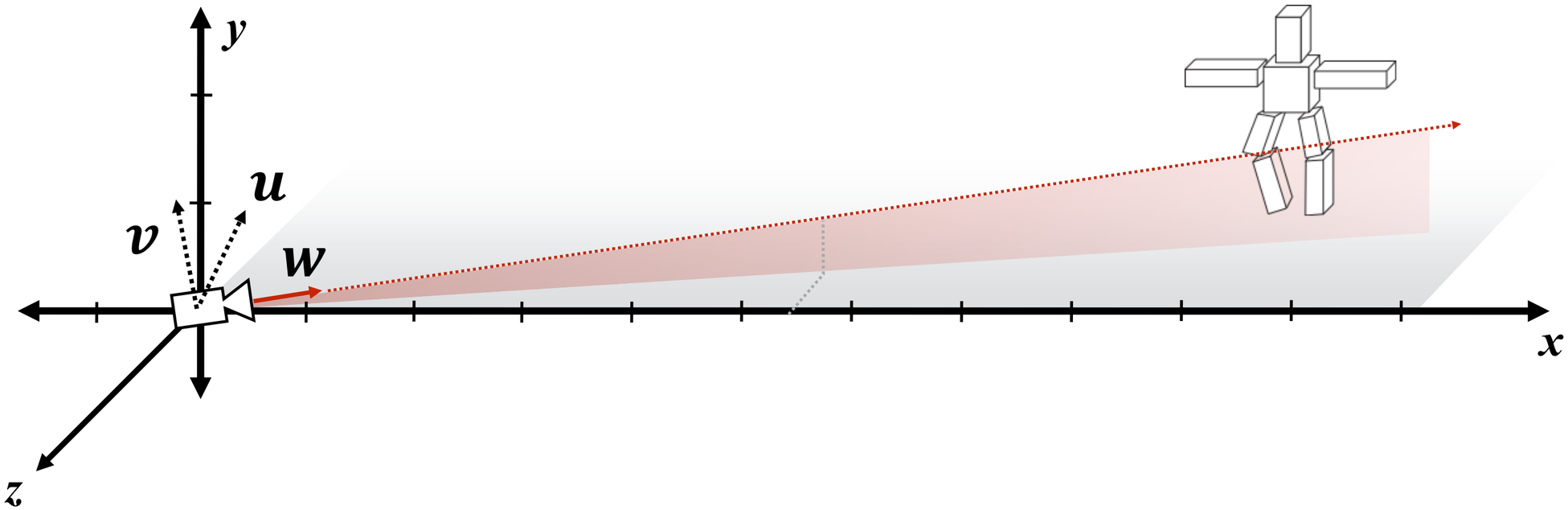
- How do you compute the transform from world to the camera coordinate system* (camera at the origin, looking down z axis)?
- Translating object vertex positions by (-4, -2, 0) yields position relative to camera.
- Rotation about y axis by $\pi/2$ gives position of object in coordinate system where camera's view direction is aligned with the -z axis

* The convenience of the camera coordinate system will become clear soon!

Camera with arbitrary orientation

Consider camera looking in direction w

What transform places the object in the camera coordinate system?



Form orthonormal basis around w: (see u and v)

Compute transformation that maps u to x-axis, v to y-axis, and w to -z axis

Answer: Consider rotation matrix (from camera to world, or world to camera?)

$$\mathbf{R} = \begin{bmatrix} \mathbf{u}_x & \mathbf{v}_x & -\mathbf{w}_x \\ \mathbf{u}_y & \mathbf{v}_y & -\mathbf{w}_y \\ \mathbf{u}_z & \mathbf{v}_z & -\mathbf{w}_z \end{bmatrix}$$

$$\mathbf{R}^T\mathbf{u} = \begin{bmatrix} \mathbf{u}\cdot\mathbf{u} & \mathbf{v}\cdot\mathbf{u} & -\mathbf{w}\cdot\mathbf{u} \end{bmatrix}^T = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$$

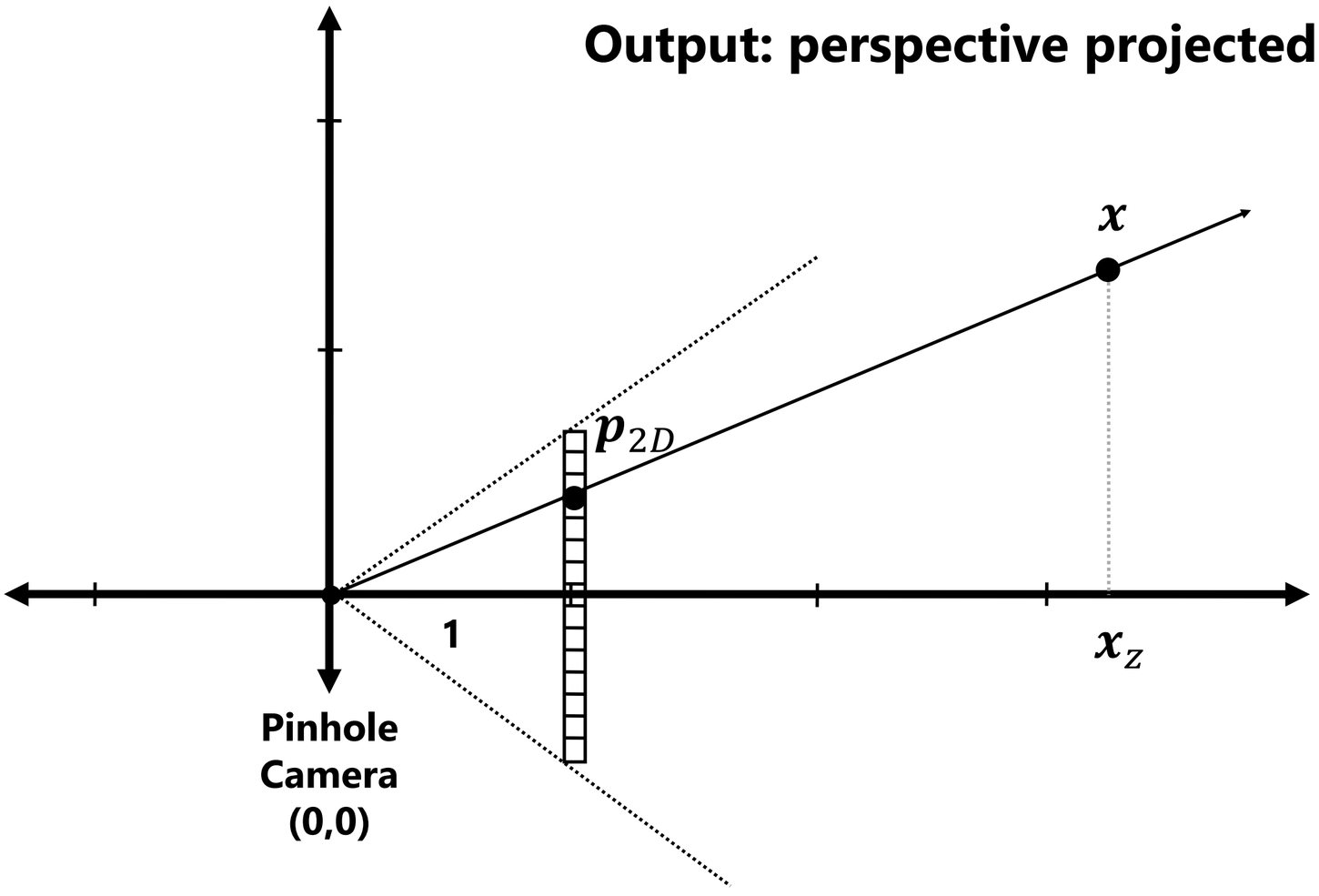
$$\mathbf{R}^T\mathbf{v} = \begin{bmatrix} \mathbf{u}\cdot\mathbf{v} & \mathbf{v}\cdot\mathbf{v} & -\mathbf{w}\cdot\mathbf{v} \end{bmatrix}^T = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$$

$$\mathbf{R}^T\mathbf{w} = \begin{bmatrix} \mathbf{u}\cdot\mathbf{w} & \mathbf{v}\cdot\mathbf{w} & -\mathbf{w}\cdot\mathbf{w} \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T$$

Basic perspective projection

Input: point in 3D $\boldsymbol{x} = (\boldsymbol{x}_x, \boldsymbol{x}_y, \boldsymbol{x}_z)$

Output: perspective projected point $\boldsymbol{p}_{2D} = (\boldsymbol{x}_x/\boldsymbol{x}_z, \boldsymbol{x}_y/\boldsymbol{x}_z)$



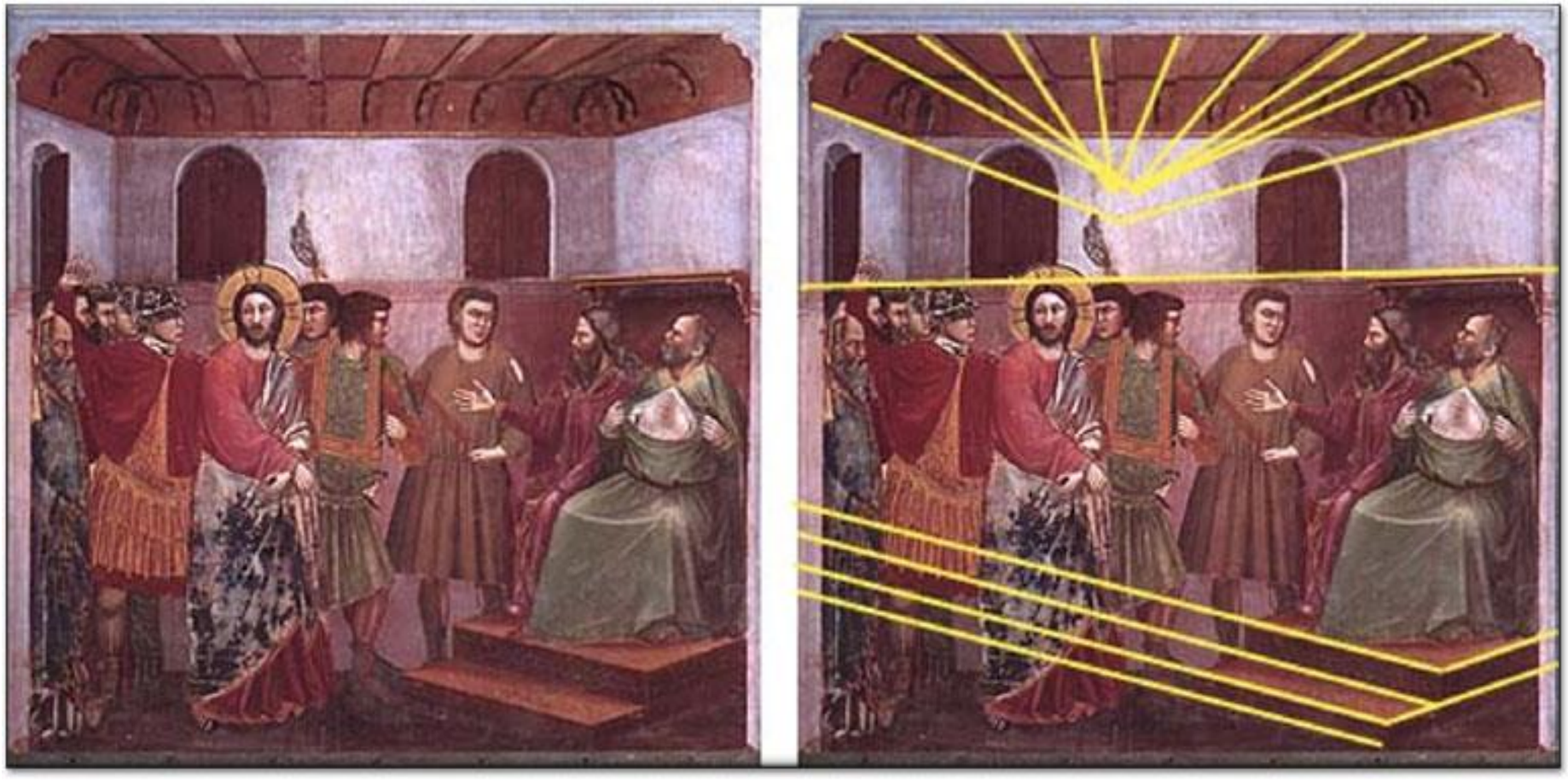
Assumption: Pinhole camera at (0,0) looking down z

Perspective projection

Early paintings: incorrect perspective

‘Jesus Before the Caïf’, by Giotto (1305)

Early paintings: incorrect perspective



‘Jesus Before the Caïf’, by Giotto (1305)

Geometrically correct perspective in art

Brunelleschi, elevation of Santo Spirito, 1434-83, Florence

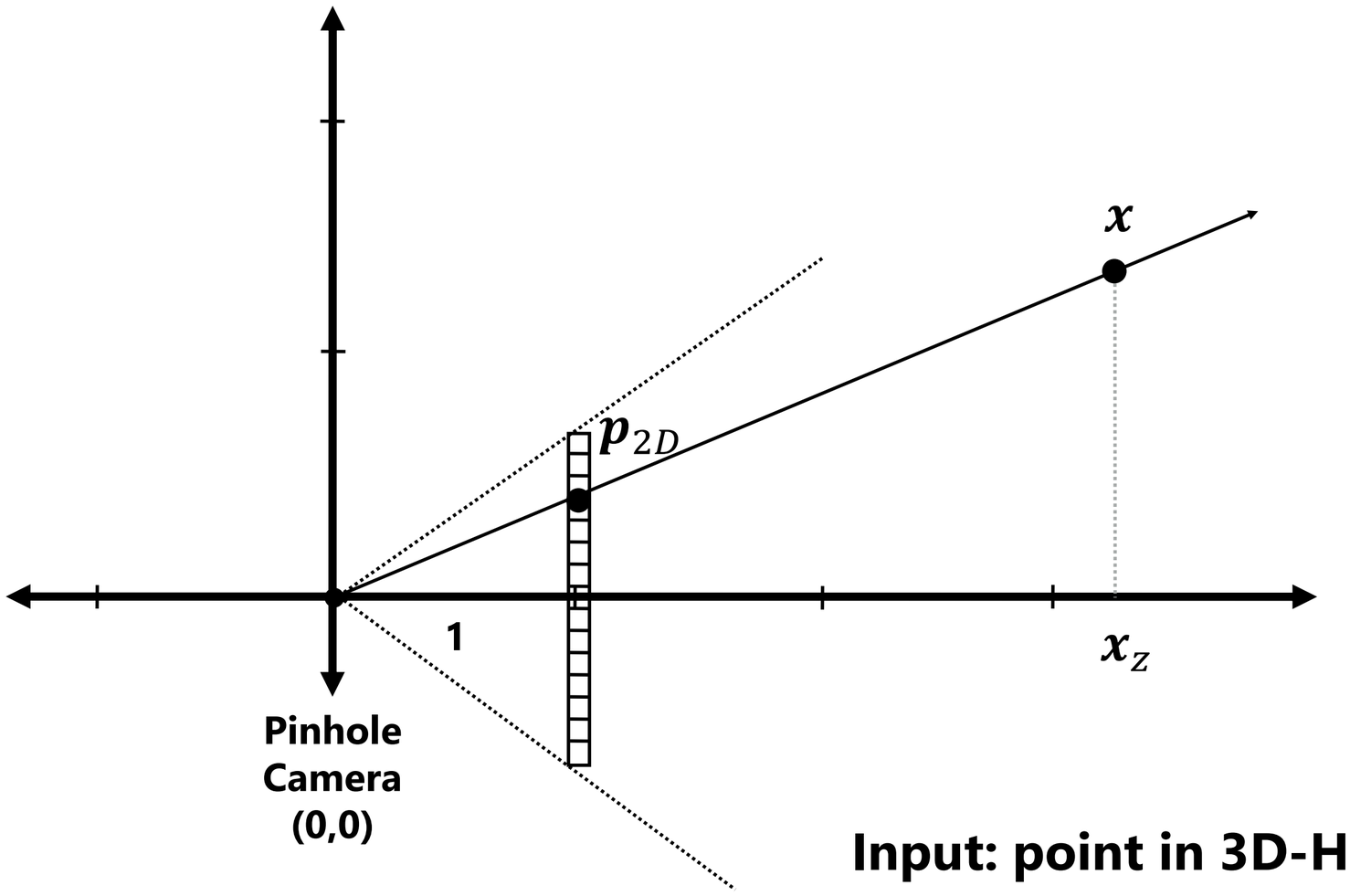
Masaccio – The Tribute Money c.1426-27 Fresco, The Brancacci Chapel, Florence

Later… rejection of proper perspective projection

Basic perspective projection

Desired perspective projected result (2D point):

$$\boldsymbol{p}_{2D} = (\boldsymbol{x}_x/\boldsymbol{x}_z, \boldsymbol{x}_y/\boldsymbol{x}_z)$$



$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Input: point in 3D-H $\boldsymbol{x} = (\boldsymbol{x}_x, \boldsymbol{x}_y, \boldsymbol{x}_z, 1)$

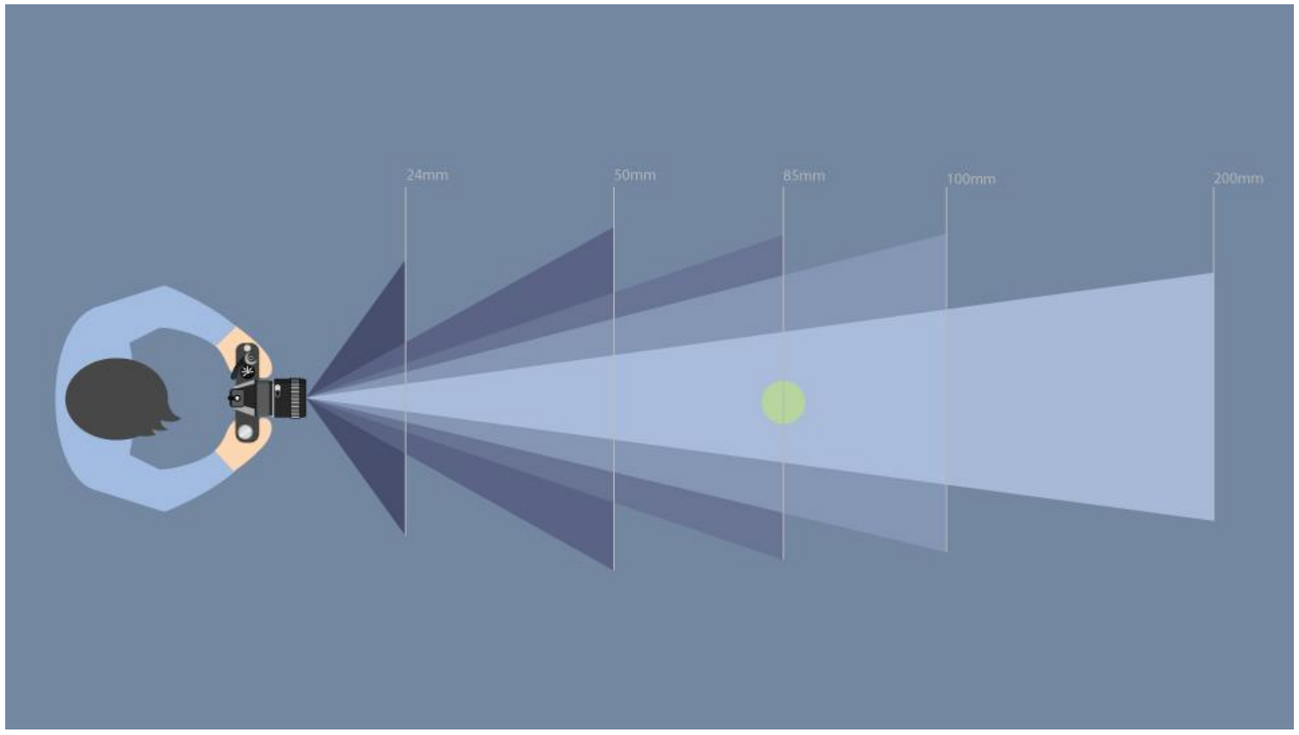
Applying map to get projected point in 3D-H $\boldsymbol{Px} = (\boldsymbol{x}_x, \boldsymbol{x}_y, \boldsymbol{x}_z, \boldsymbol{x}_z)$

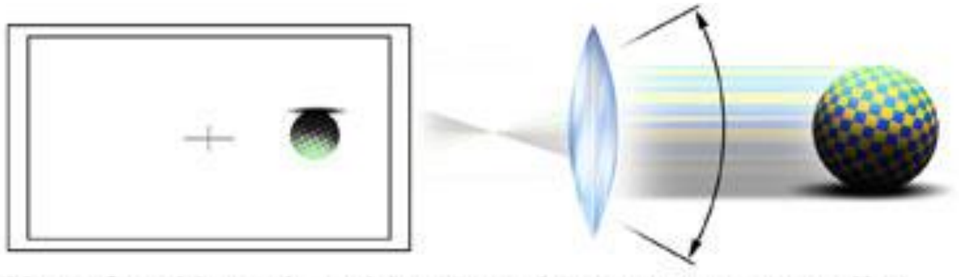
Point projected to 2D-H (drop z coord) $\boldsymbol{p}_{2D-H} = (\boldsymbol{x}_x, \boldsymbol{x}_y, \boldsymbol{x}_z)$

Point in 2D (homogeneous divide) $\boldsymbol{p}_{2D} = (\boldsymbol{x}_x/\boldsymbol{x}_z, \boldsymbol{x}_y/\boldsymbol{x}_z)$

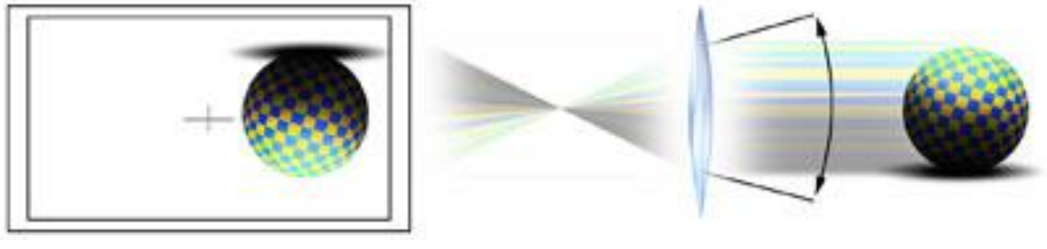
Assumption: Pinhole camera at (0,0) looking down z

Changing the shape of the view frustum



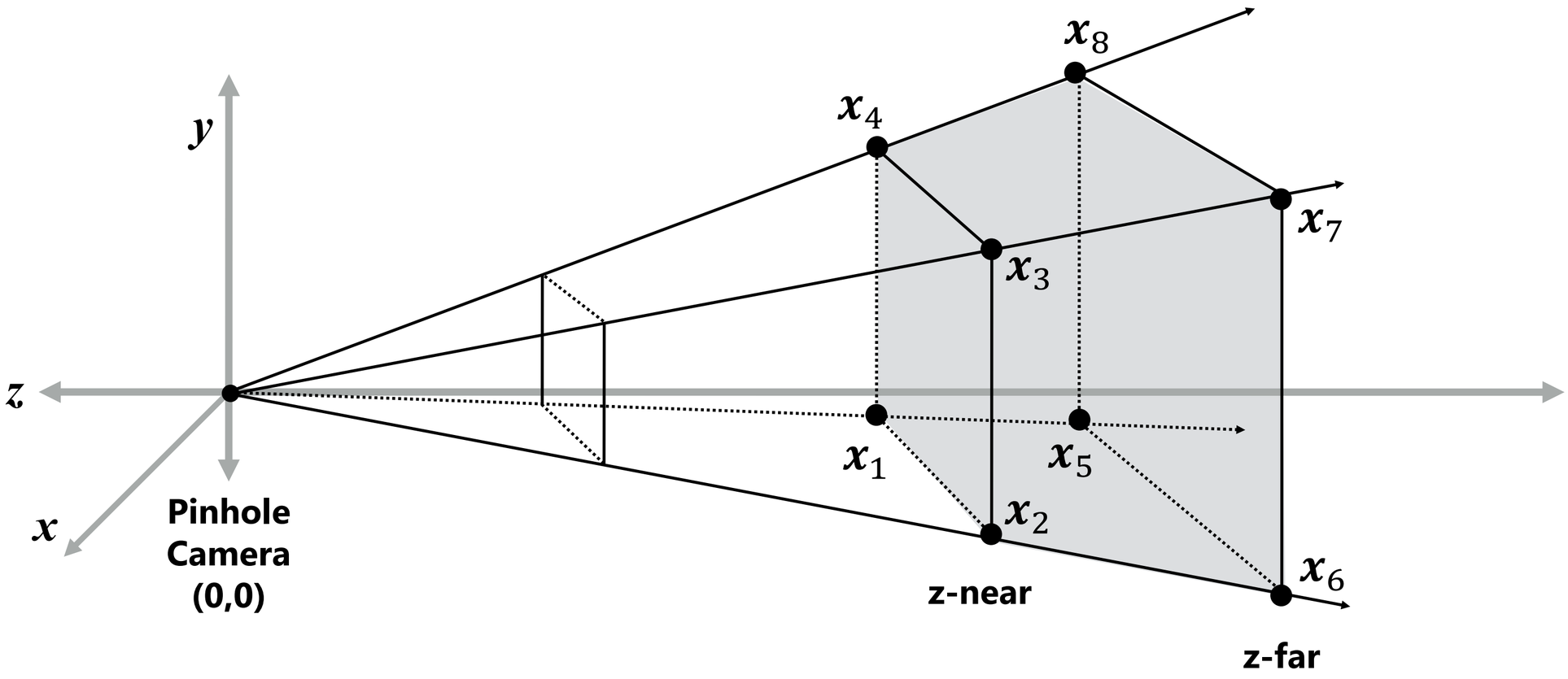


Short focal length - wide FOV, objects appear smaller.



Long focal length - narrow FOV, objects appear larger.

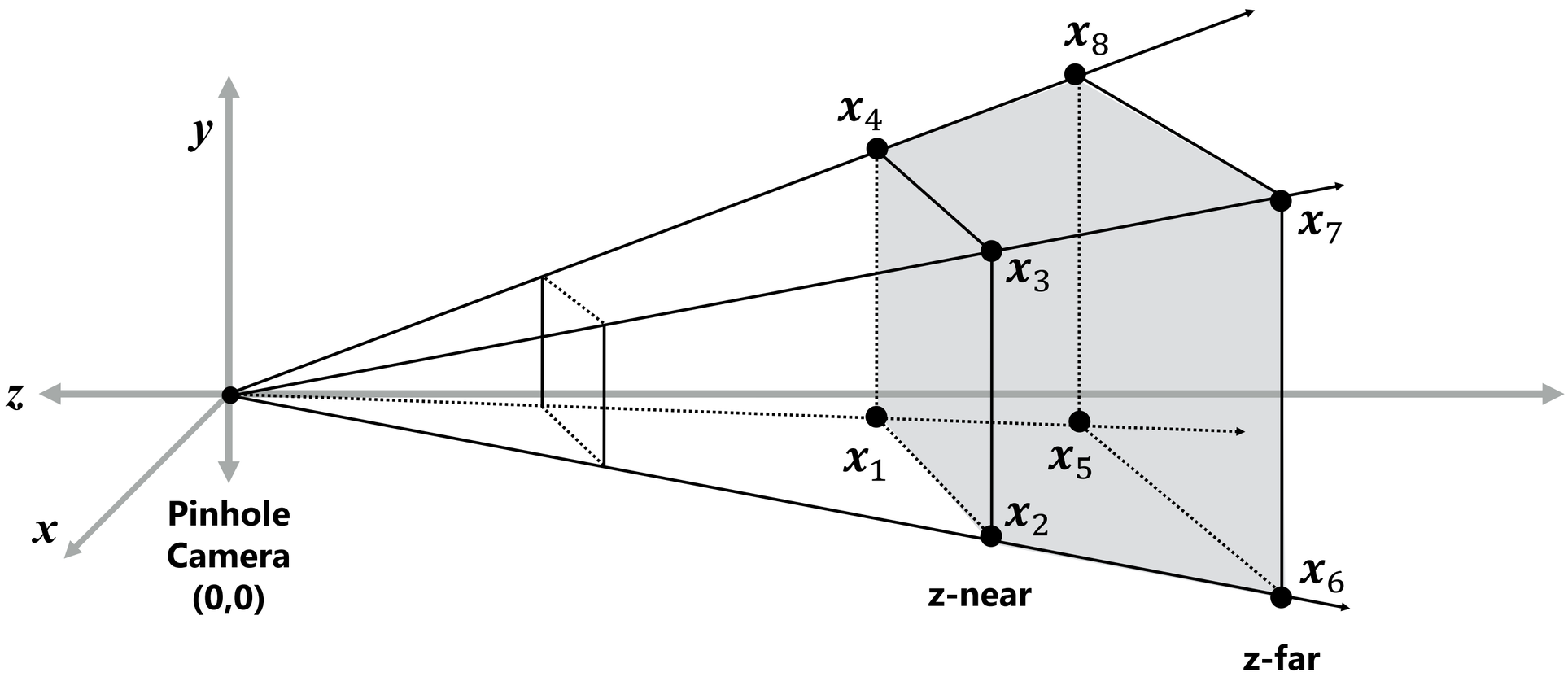
The view frustum



View frustum: region in space that will appear on the screen

How can we change the shape of the view frustum?

The view frustum

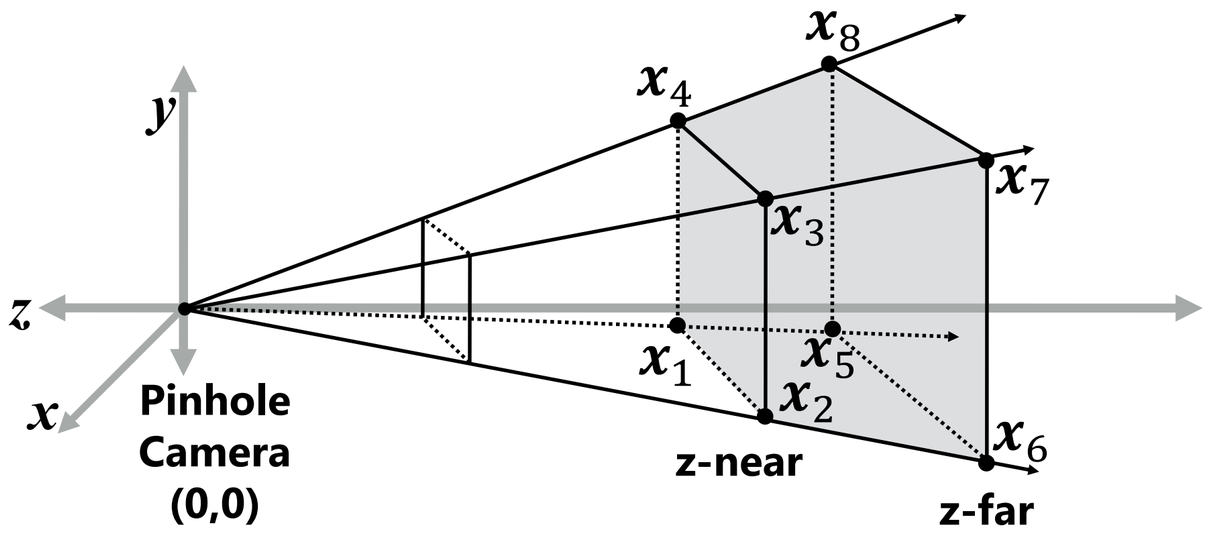


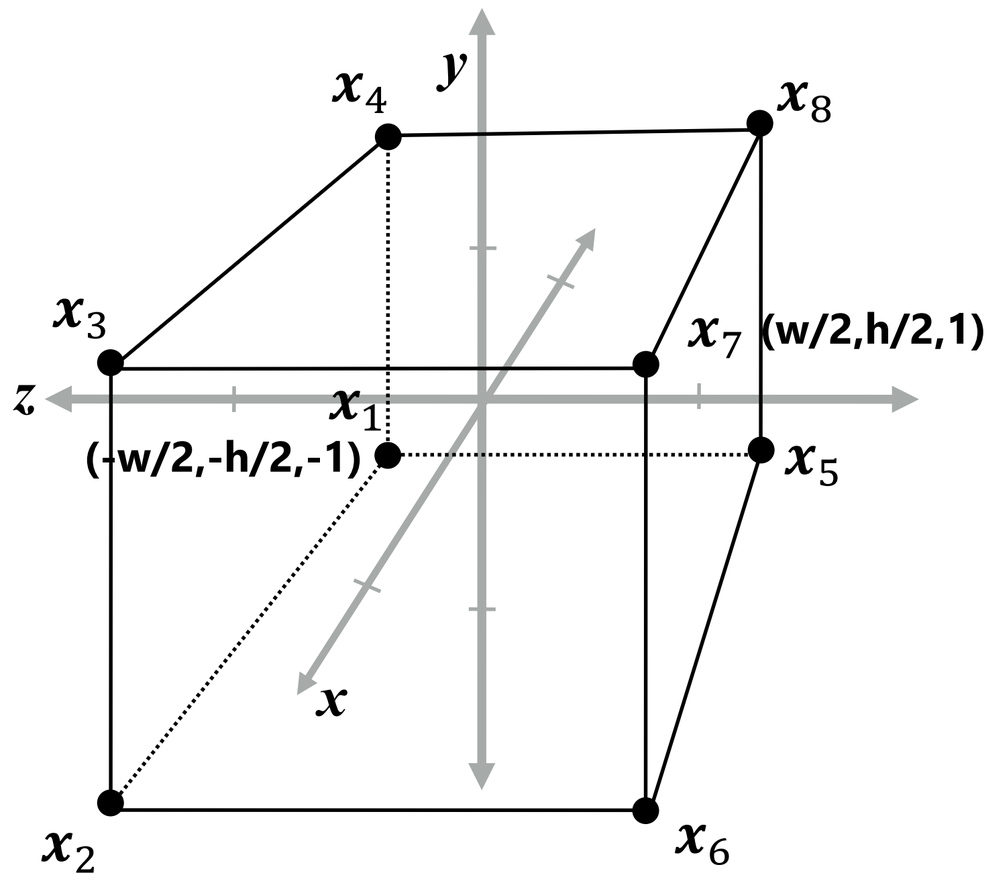
View frustum: region in space that will appear on the screen

How can we change the shape of the view frustum?

But first, want a transformation that maps view frustum to a unit cube (computing screen coordinates in that space is then trivial)

The view frustum





Projecting on virtual screen

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{Px} = (\boldsymbol{x}_x, \boldsymbol{x}_y, \boldsymbol{x}_z, \boldsymbol{x}_z)$$

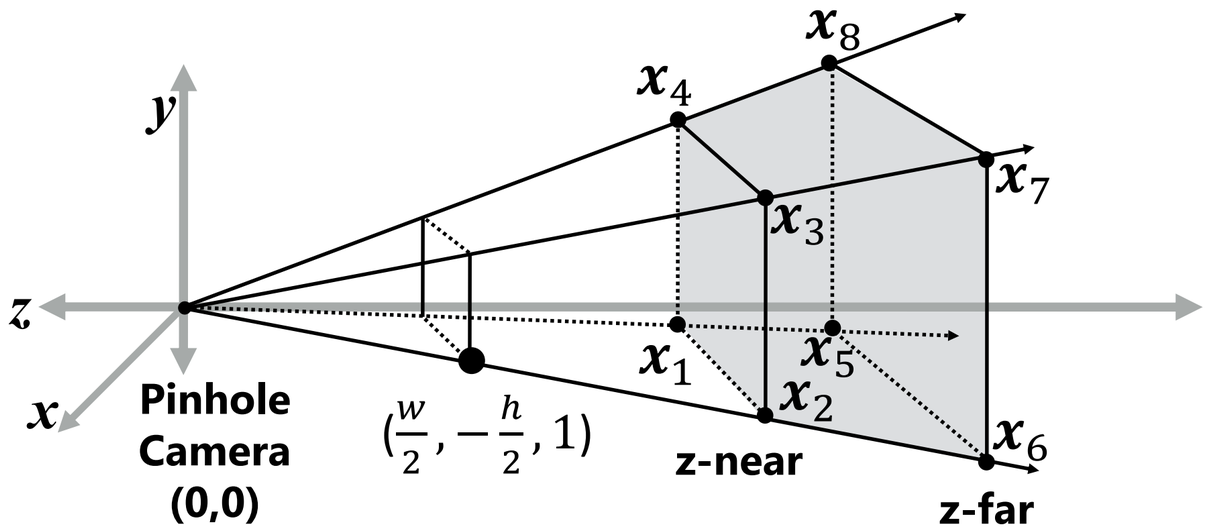
First step: apply perspective transform & normalize z-coord
What are the values of a and b?

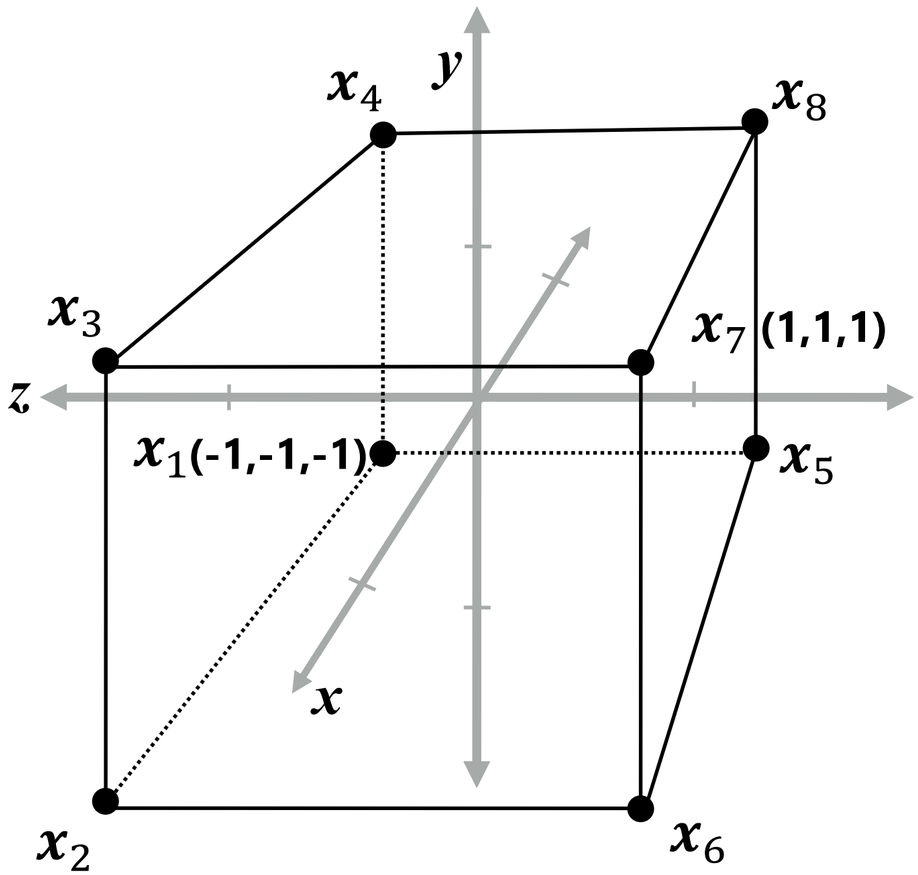
$$\mathbf{P}_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{x}_1 = (\boldsymbol{x}_{1x}, \boldsymbol{x}_{1y}, -zNear, 1)\ ;\ \boldsymbol{Px}_1 = (\boldsymbol{x}_{1x}, \boldsymbol{x}_{1y}, -zNear, zNear)$$

$$\boldsymbol{x}_7 = (\boldsymbol{x}_{7x}, \boldsymbol{x}_{7y}, -zFar, 1);\ \boldsymbol{Px}_7 = (\boldsymbol{x}_{7x}, \boldsymbol{x}_{7y}, zFar, zFar)$$

The view frustum





Changing the shape of the view frustum:

θ**: field of view in y direction (**$h = 2tan\,(\frac{\theta}{2})$**)**

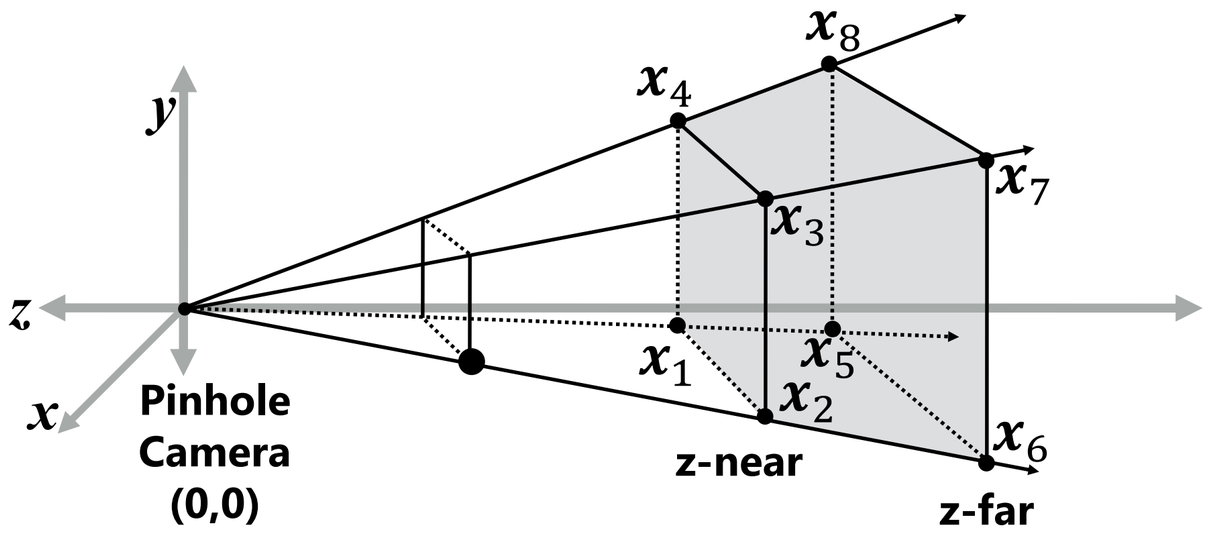
r = ***aspect ratio*** **= width / height**

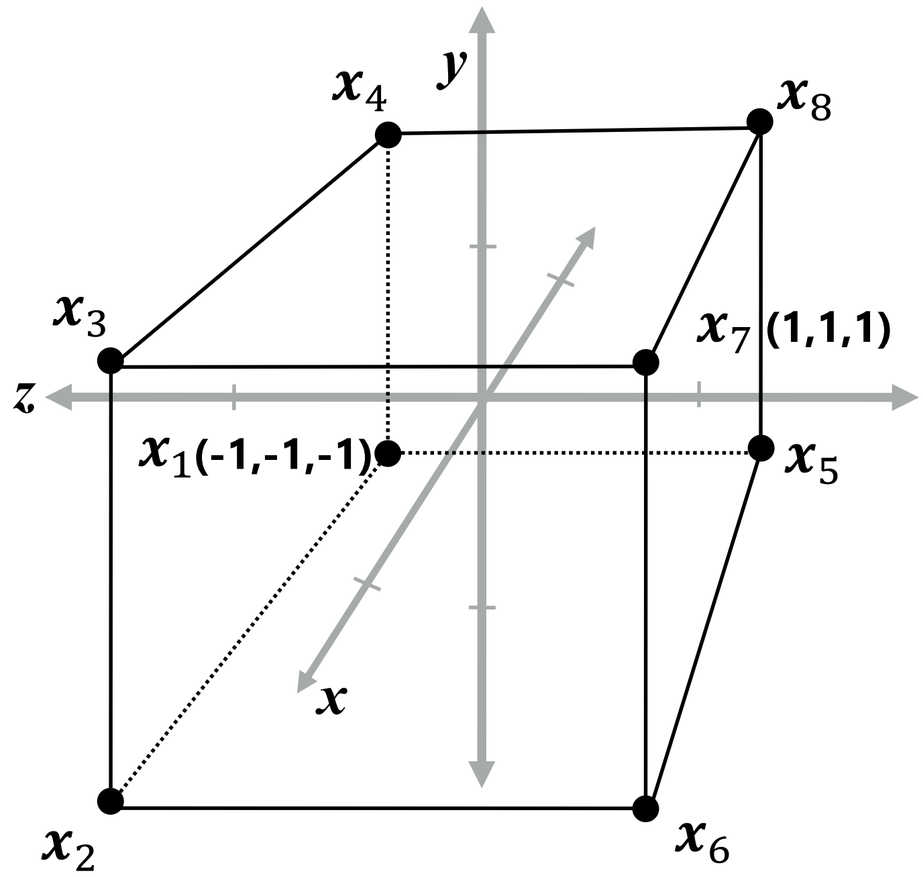
Rescale x and y components such that frustum maps to unit cube

$$\begin{bmatrix} f/r & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$f = \boldsymbol{cot}(\frac{\boldsymbol{\theta}}{\boldsymbol{2}})$$

The view frustum



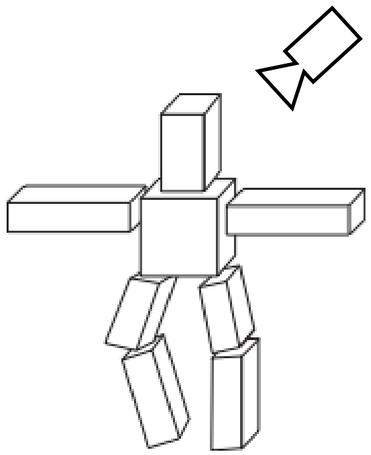


Putting it all together, transformation matrix that maps view frustum to unit cube:

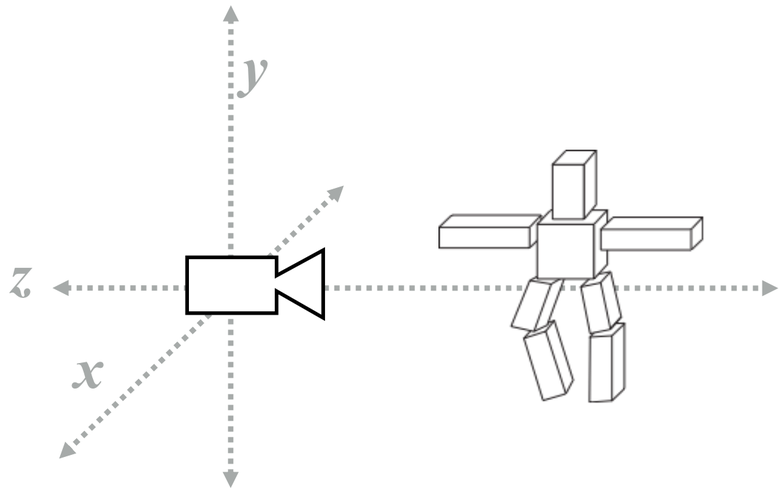
$$\mathbf{P} = \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{zfar+znear}{znear-zfar} & \frac{2\times zfar\times znear}{znear-zfar} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Is the perspective projection matrix invertible? What does that mean?

Transformations recap

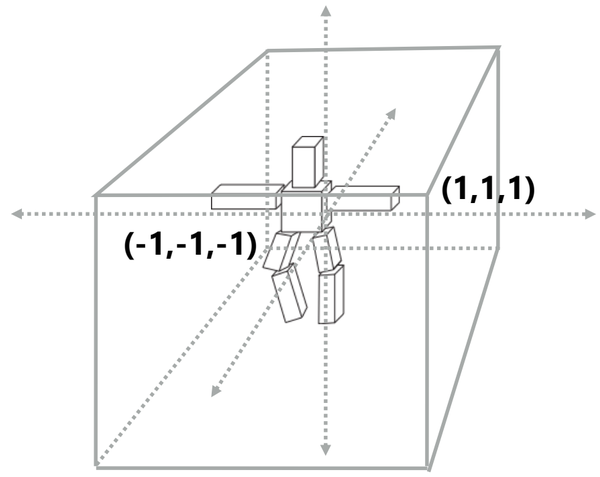


Modeling transforms:
Position object in scene



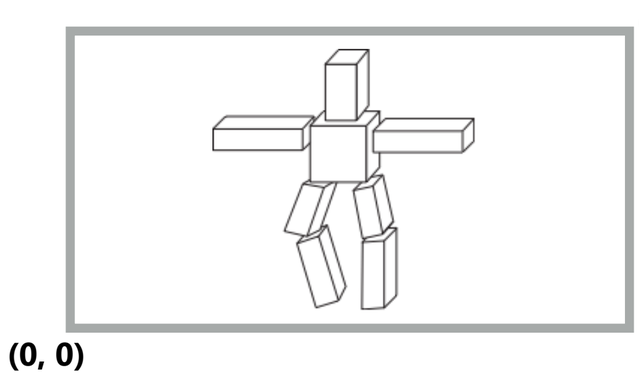
Viewing (camera) transform:
positions objects in coordinate space relative to camera

Canonical form: camera at origin looking down -z



Projection transform + homogeneous divide:
Performs perspective projection

Canonical form: visible region of scene contained within unit cube



Screen transform:
objects now in 2D screen coordinates

Review exercise: screen transform *

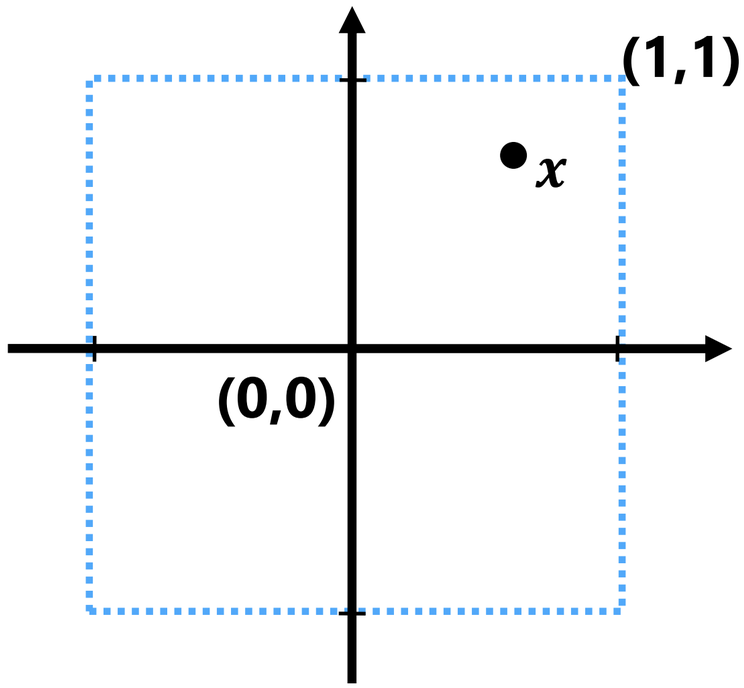
Convert points in normalized coordinate space to screen pixel coordinates

Example:

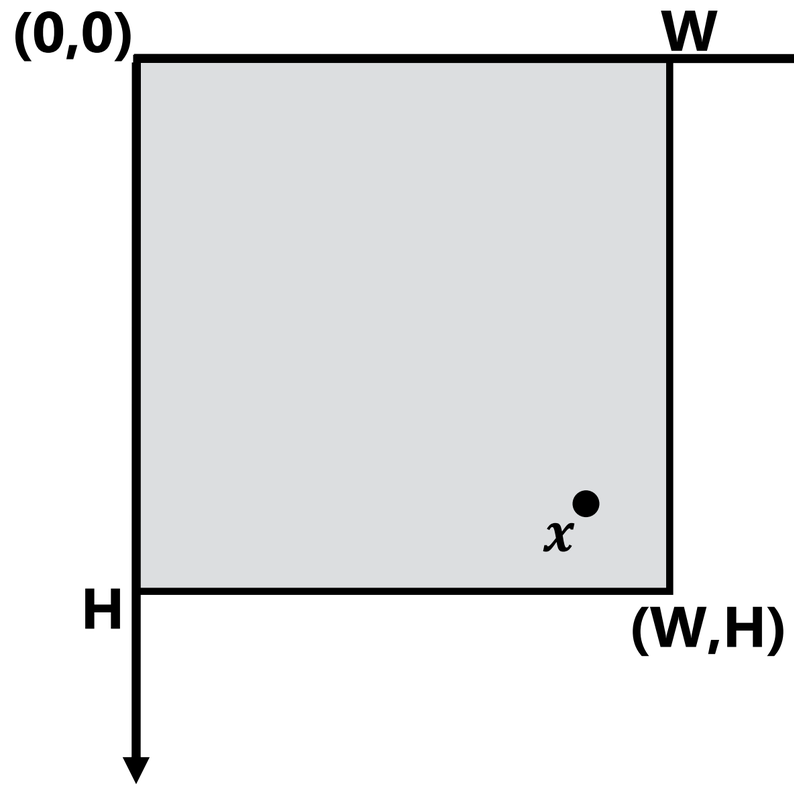
All points within (-1,1) to (1,1) region are on screen

(1,1) in normalized space maps to (W,0) in screen

Normalized coordinate space:



Screen (W x H output image) coordinate space:



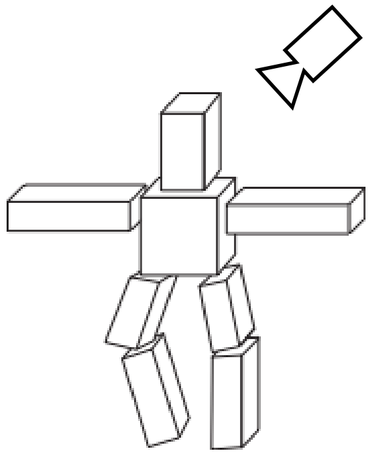
Step 1: reflect about x

Step 2: translate by (1,1)

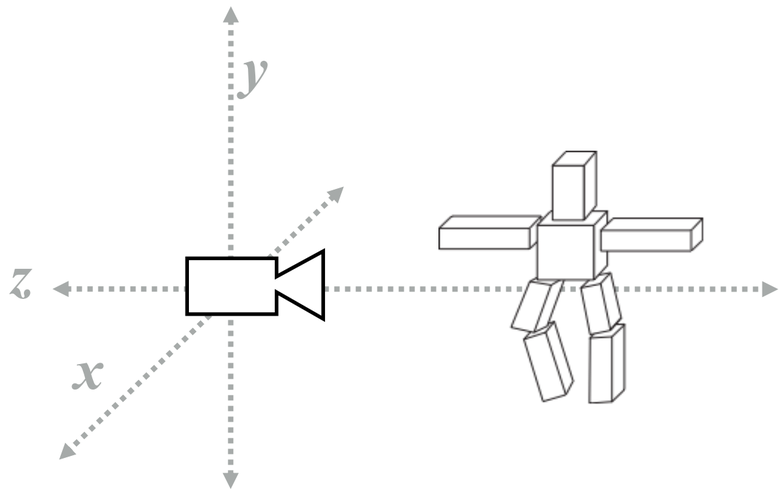
Step 3: scale by (W/2,H/2)

Many 3D graphics systems like OpenGL place (0,0) in bottom-left. In this case what would the transform be?

Transformations recap

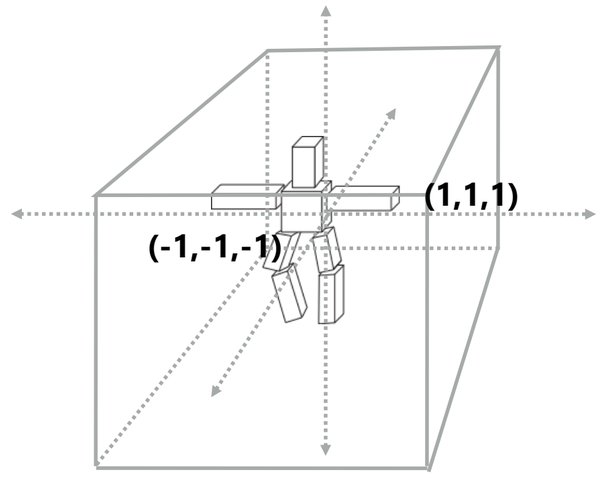


Modeling transforms:
Position object in scene



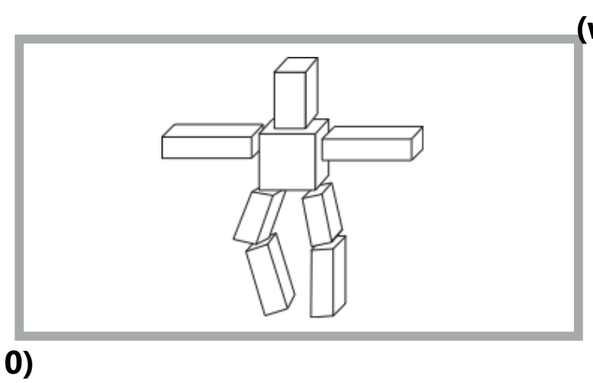
Viewing (camera) transform:
positions objects in coordinate space relative to camera

Canonical form: camera at origin looking down -z



Projection transform +
homogeneous divide:
Performs perspective projection

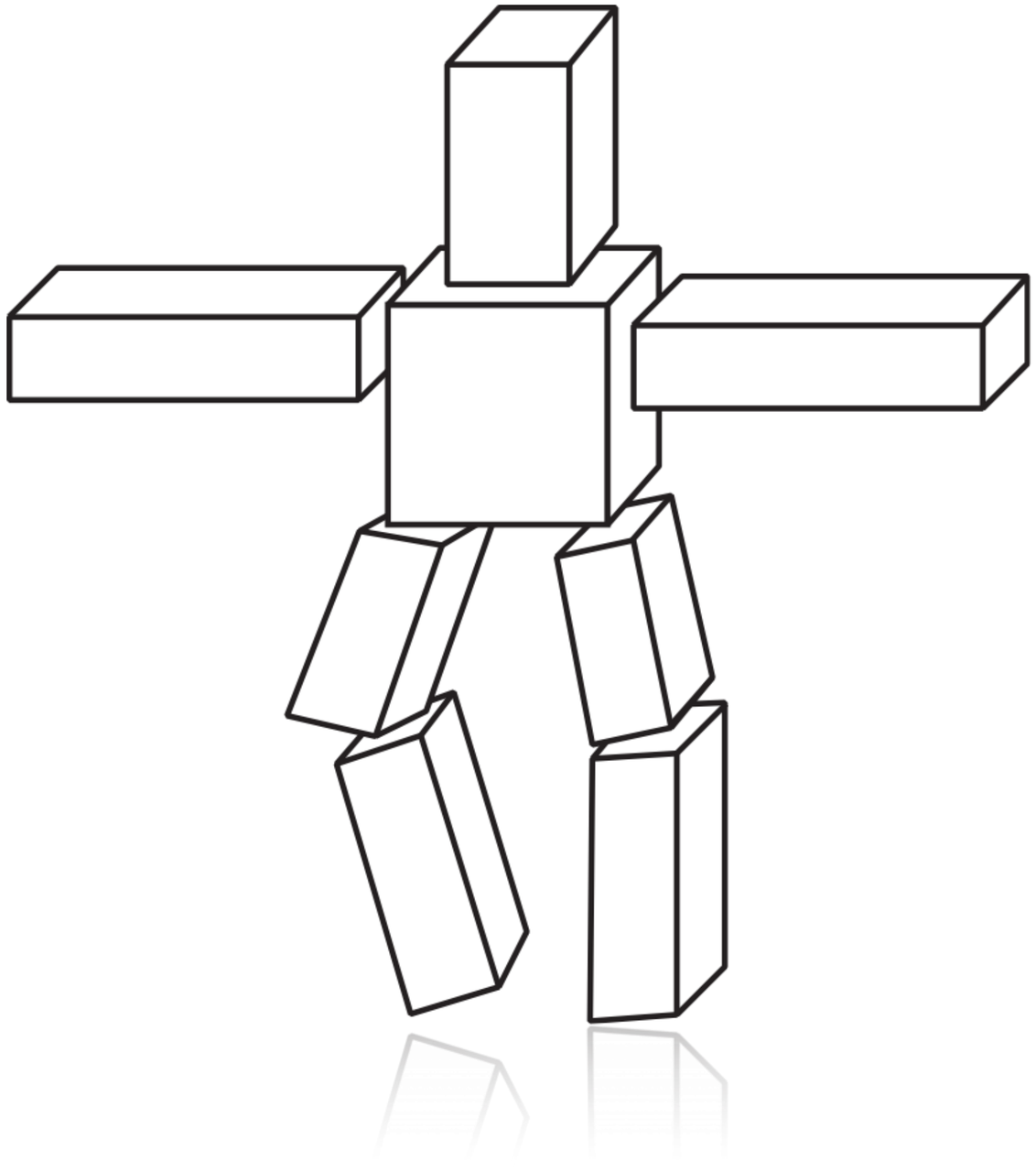
Canonical form: visible region of scene contained within unit cube



Screen transform:
objects now in 2D screen coordinates

Compute
screen coverage from 2D object position

But how do we go beyond cube-man?



How do we get this rich “geometry”?

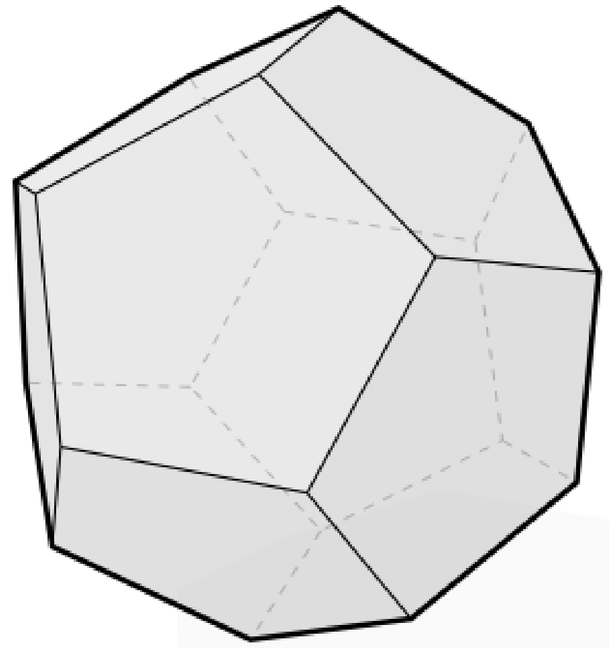
What is geometry?

What is geometry?

“Earth” “measure”

ge•om•et•ry /jēˈämətrē/ *n.*

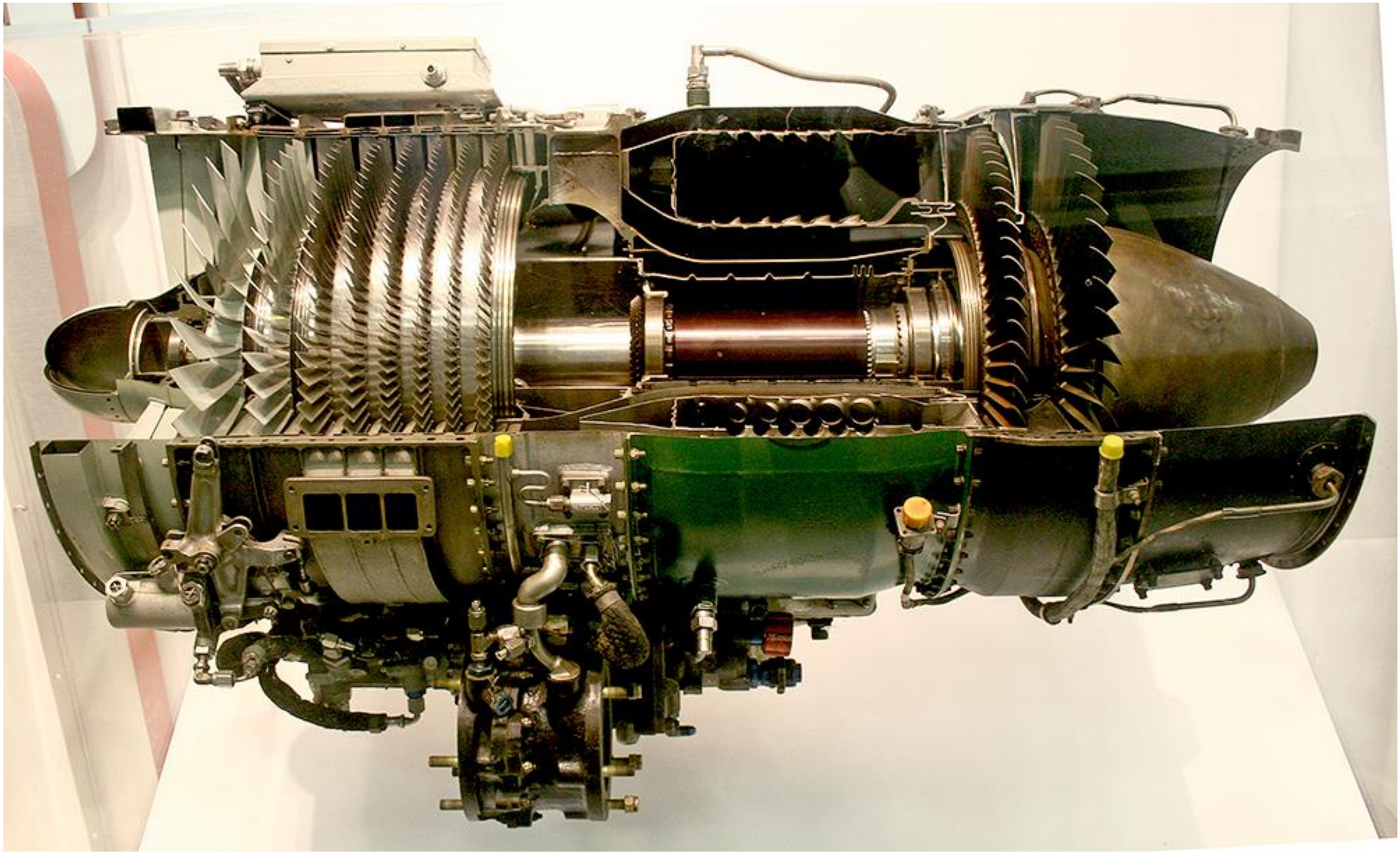
1. The study of shapes, sizes, patterns, and positions.
2. The study of spaces where some quantity (lengths, angles, etc.) can be *measured*.



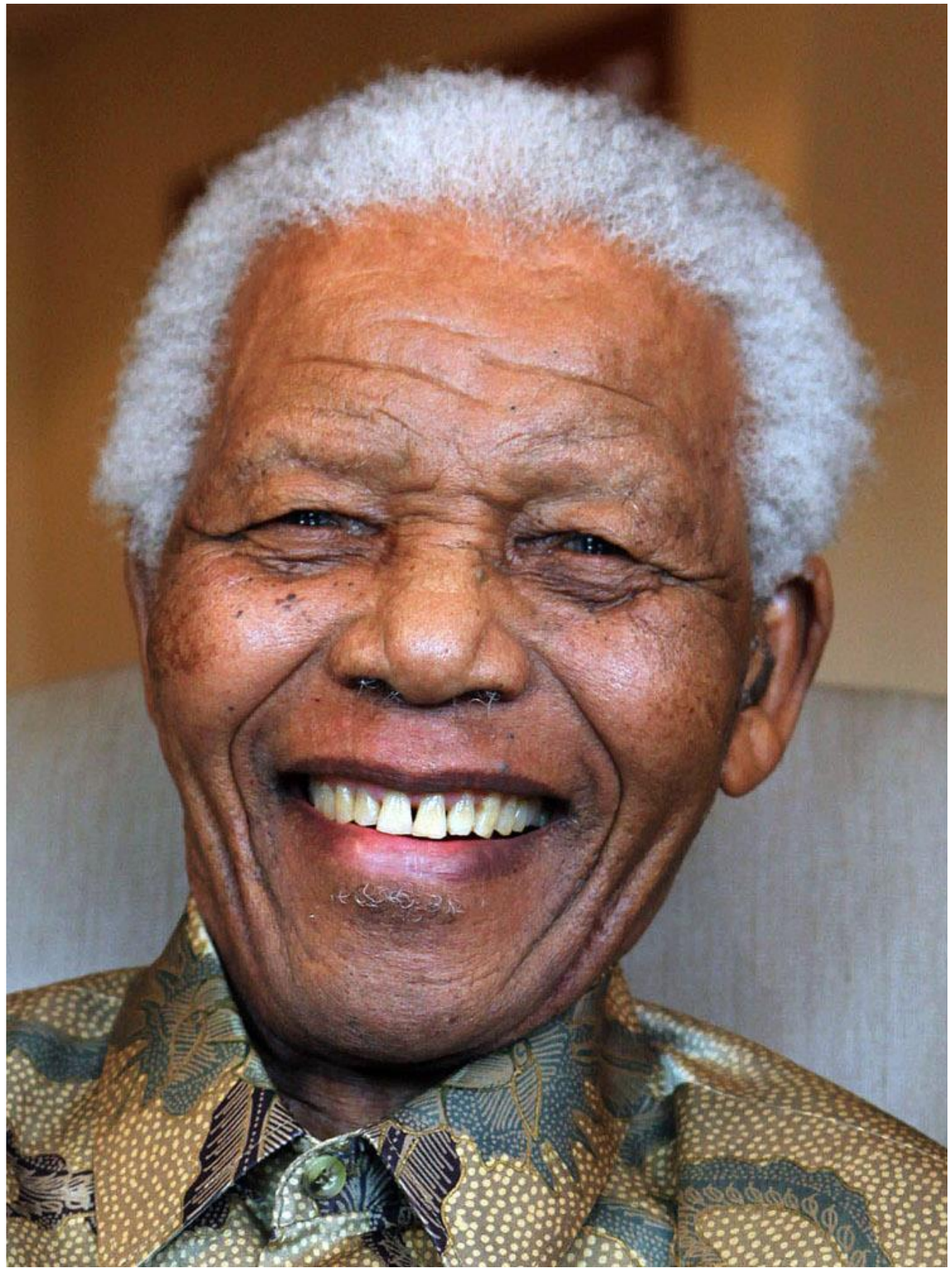
Plato: “...the earth is in appearance like one of those balls which have leather coverings in twelve pieces...”

Examples of geometry

Examples of geometry

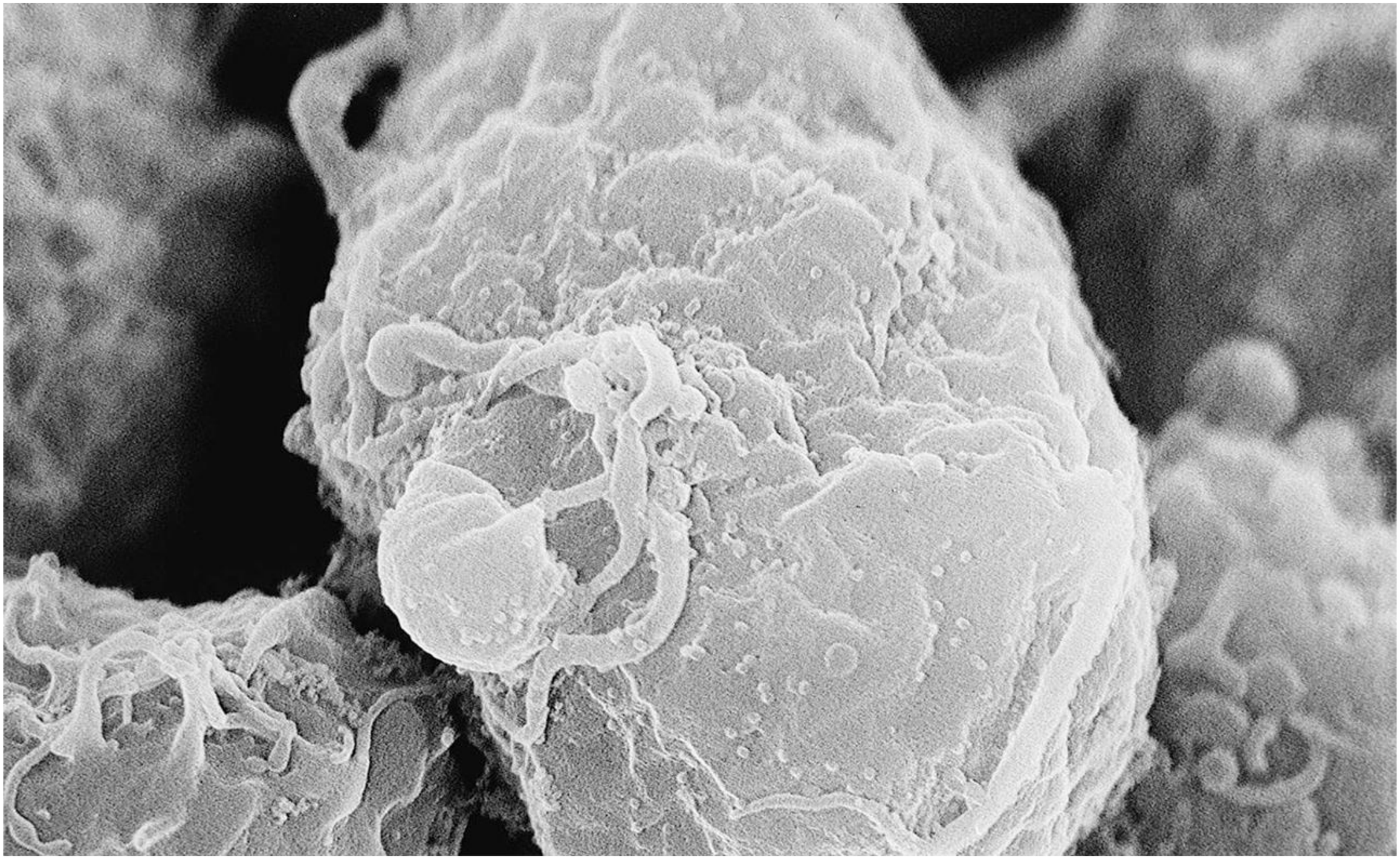


Examples of geometry



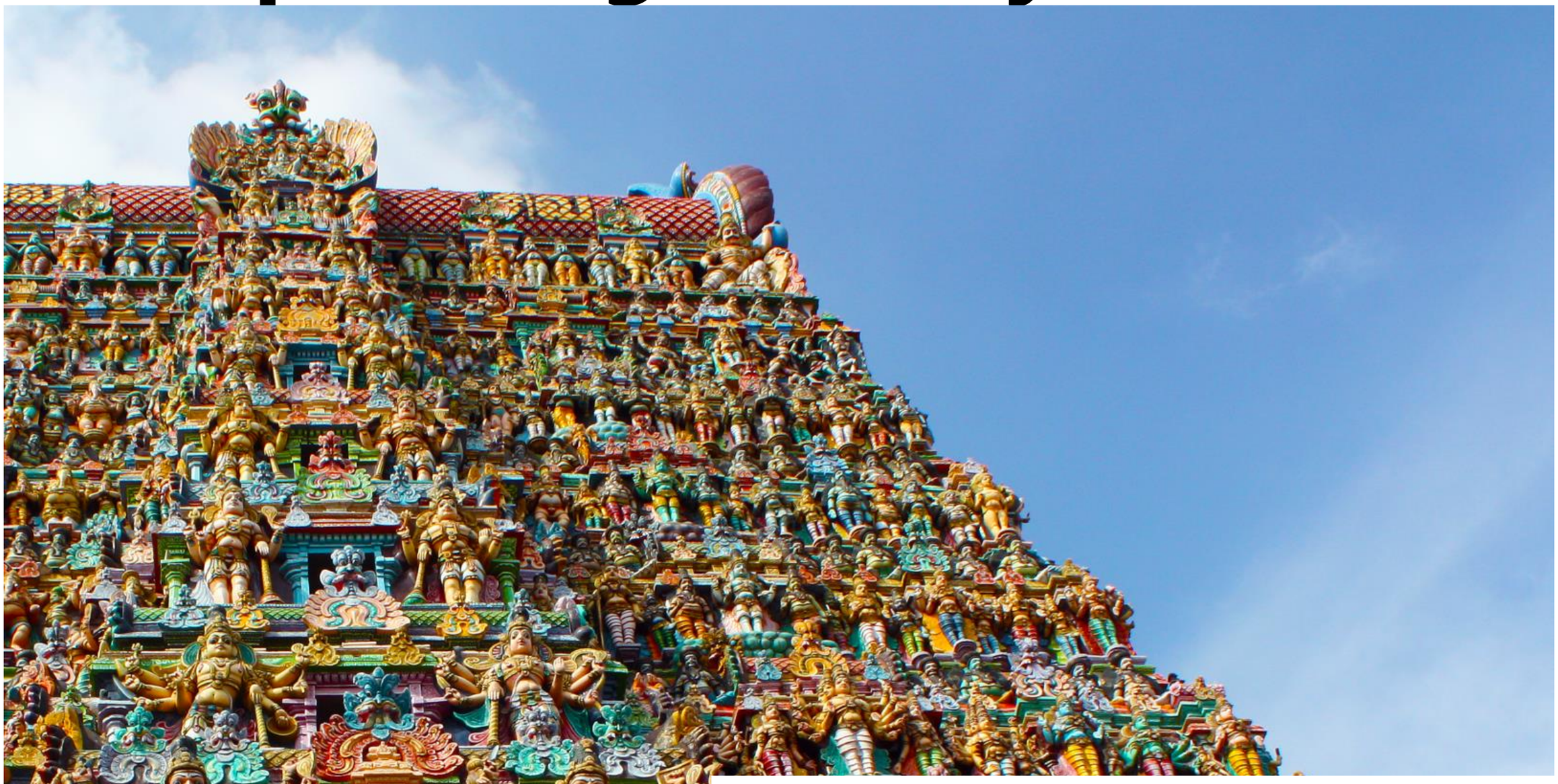
Examples of geometry

Examples of geometry



Examples of geometry

Examples of geometry



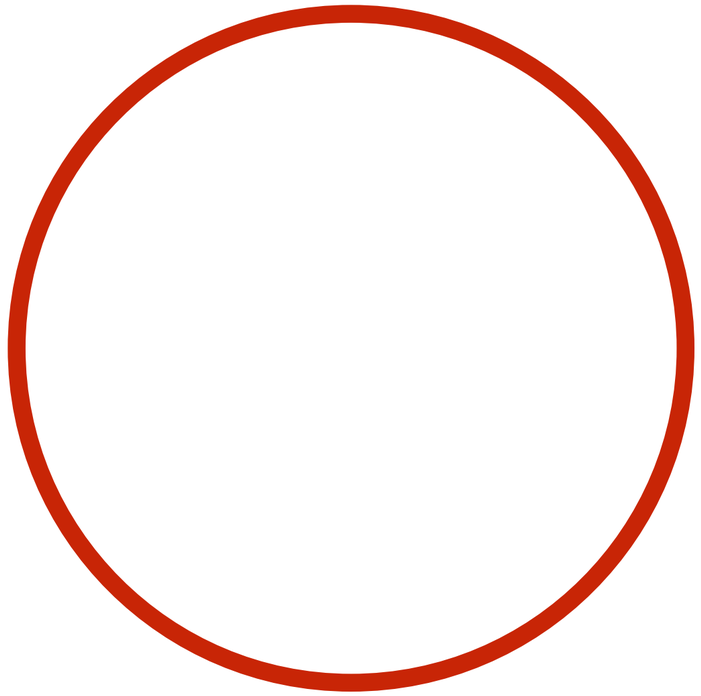
How can we describe geometry?

IMPLICIT

$$x^2+y^2=1$$

LINGUISTIC

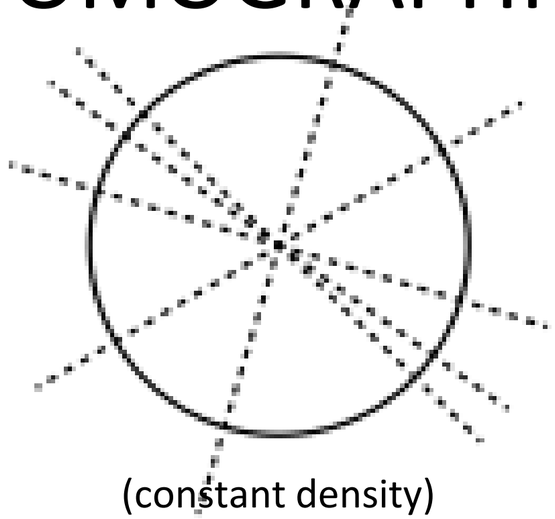
“unit circle”



EXPLICIT

$$(\underbrace{\cos\theta}_{x}, \underbrace{\sin\theta}_{y})$$

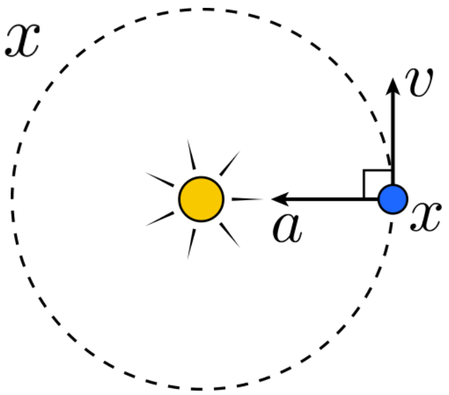
TOMOGRAPHIC



(constant density)

DYNAMIC

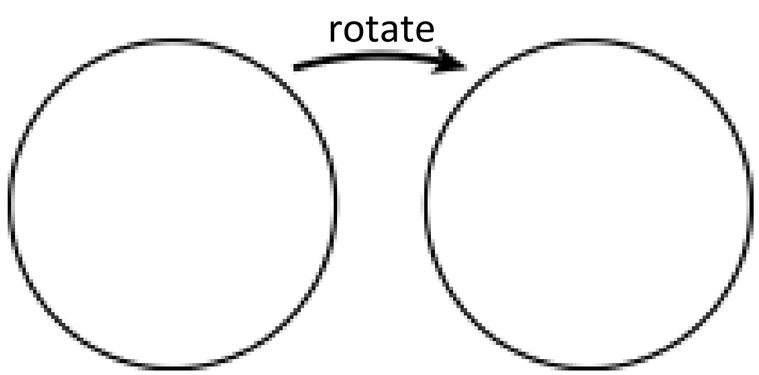
$$\frac{d^2}{dt^2}x = -x$$



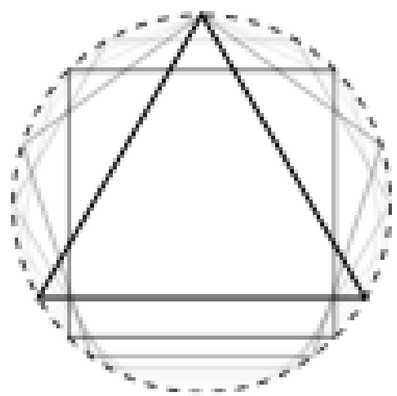
CURVATURE

$$\kappa = 1$$

SYMMETRIC



DISCRETE



n → ∞

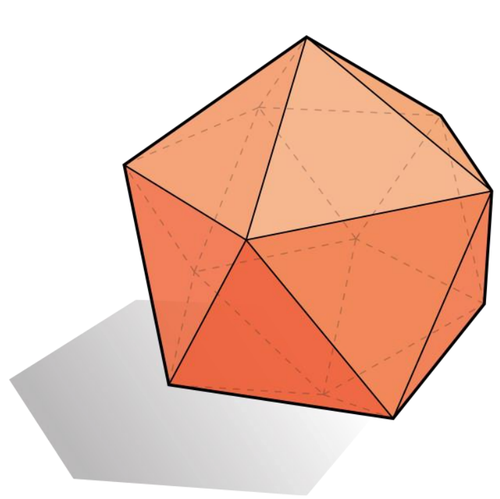
Given all these options, what's the best way to encode geometry on a computer?

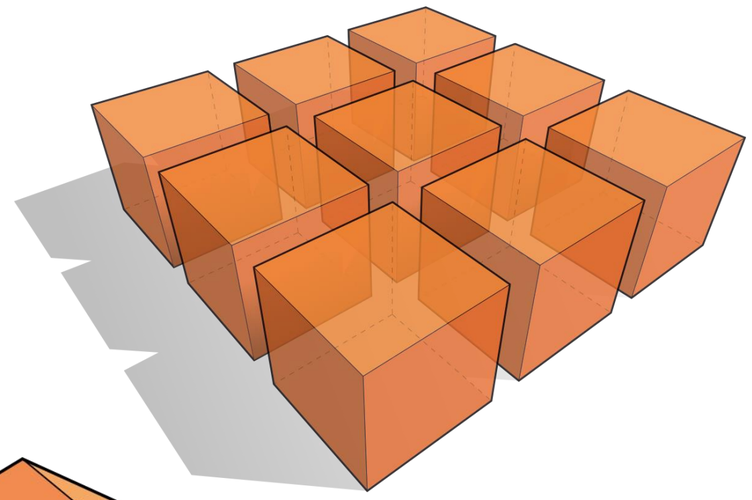
No one "best" choice.

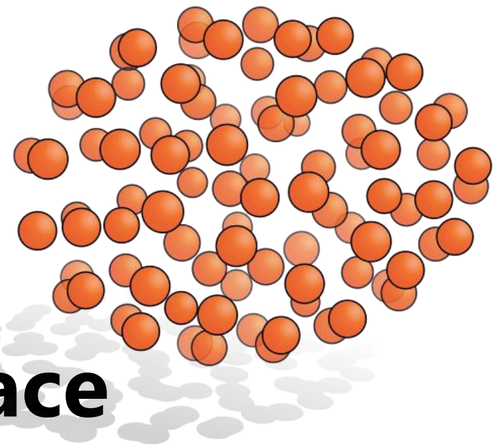
Geometry is hard!

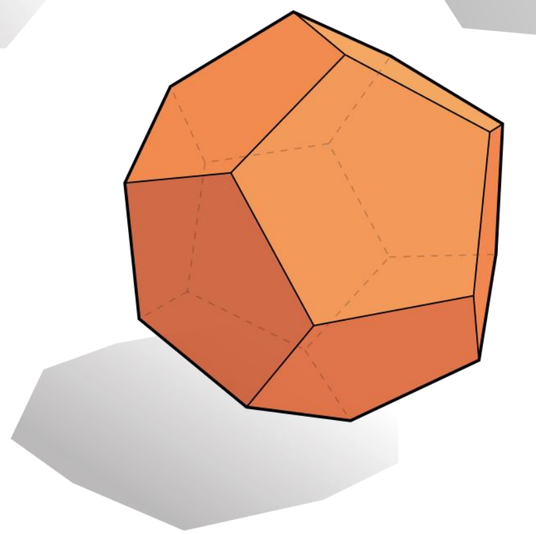
Many ways to digitally encode geometry

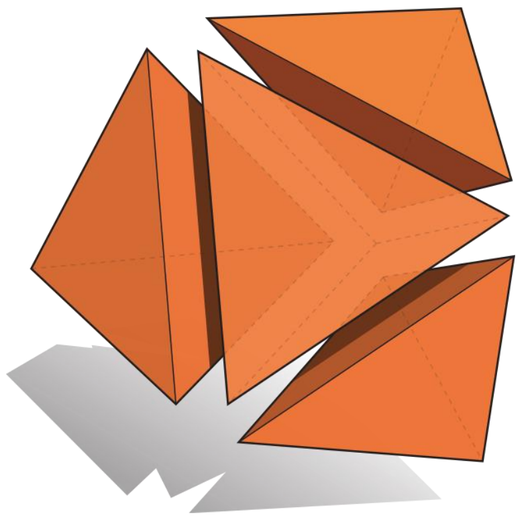
- **EXPLICIT**
  - **point cloud**
  - **polygon mesh**
  - **subdivision, NURBS**
  - **…**
- **IMPLICIT**
  - **level set**
  - **algebraic surface**
  - **L-systems**
  - **…**







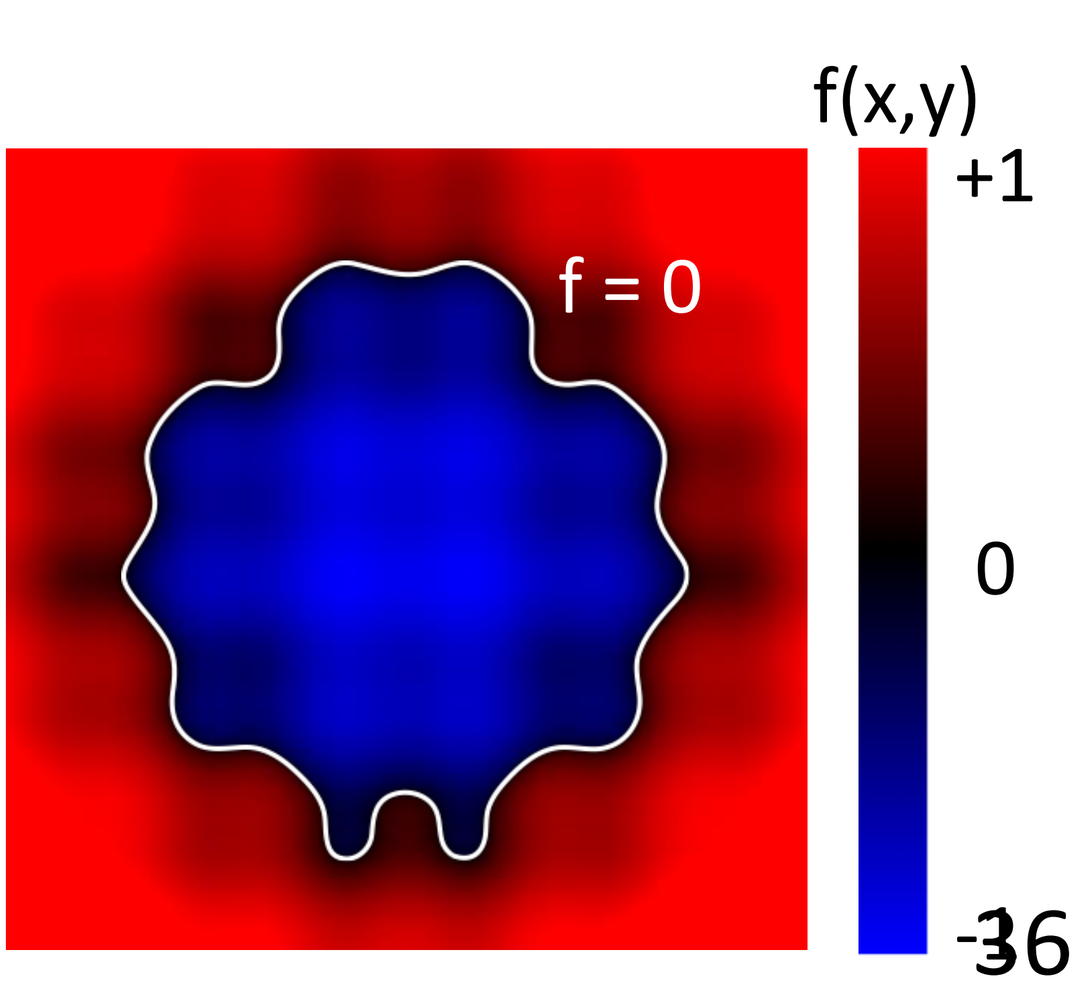




- **Each choice best suited to a different task/type of geometry**

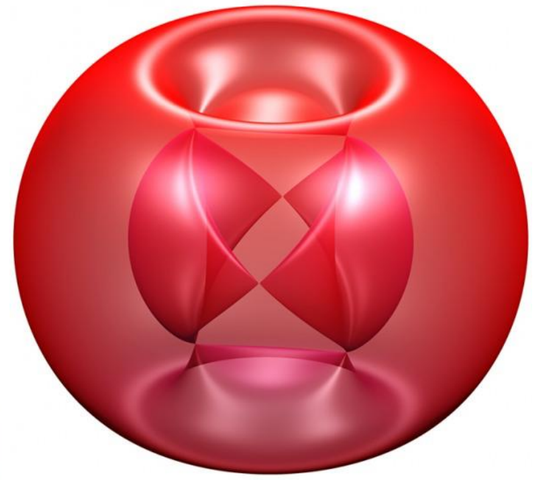
“Implicit” Representations of Geometry

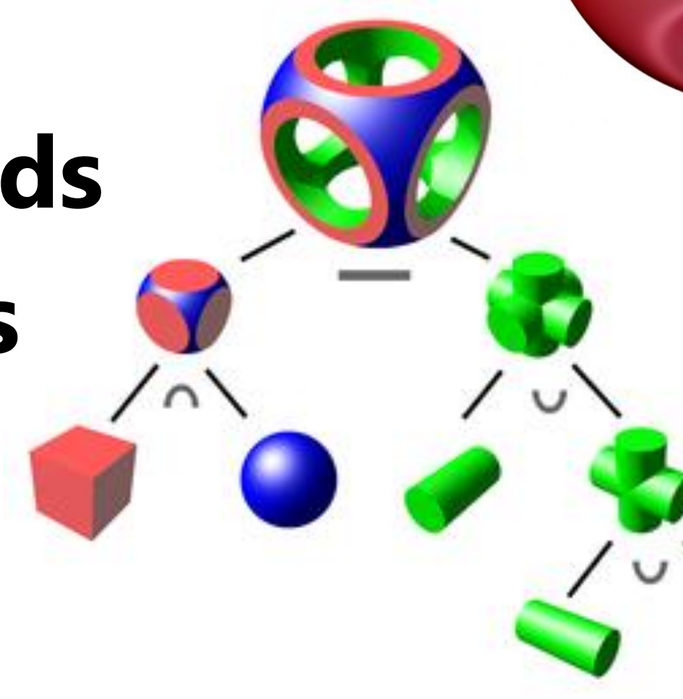
- Points aren’t known directly, but satisfy some relationship
- E.g., unit sphere is all points x such that $x^2+y^2+z^2=1$
- More generally, f(x,y,z) = 0

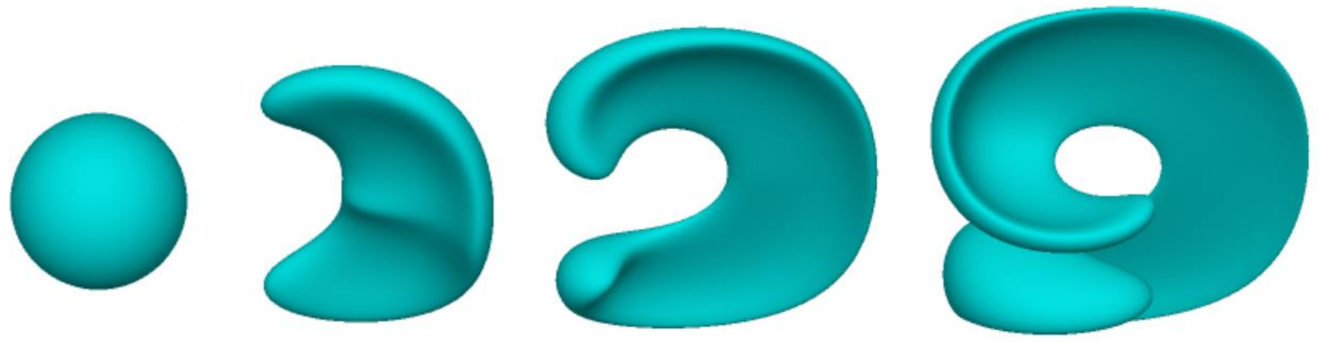


Many implicit representations in graphics

- **algebraic surfaces**
- **constructive solid geometry**
- **level set methods**
- **blobby surfaces**
- **fractals**
- **…**







(Will see some of these a bit later.)

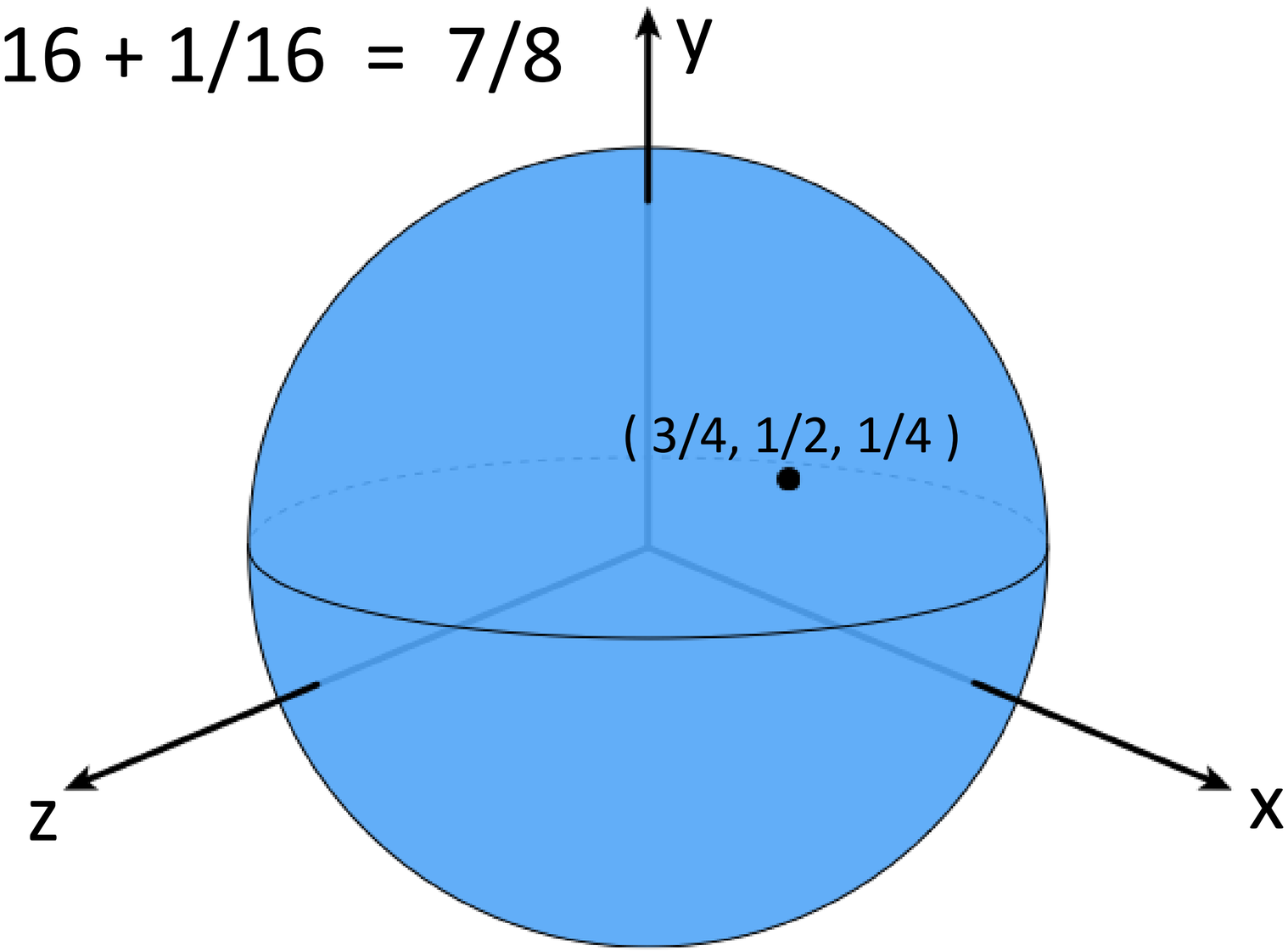
Check if a point is inside the unit sphere

How about the point (3/4, 1/2, 1/4)?

9/16 + 4/16 + 1/16 = 7/8

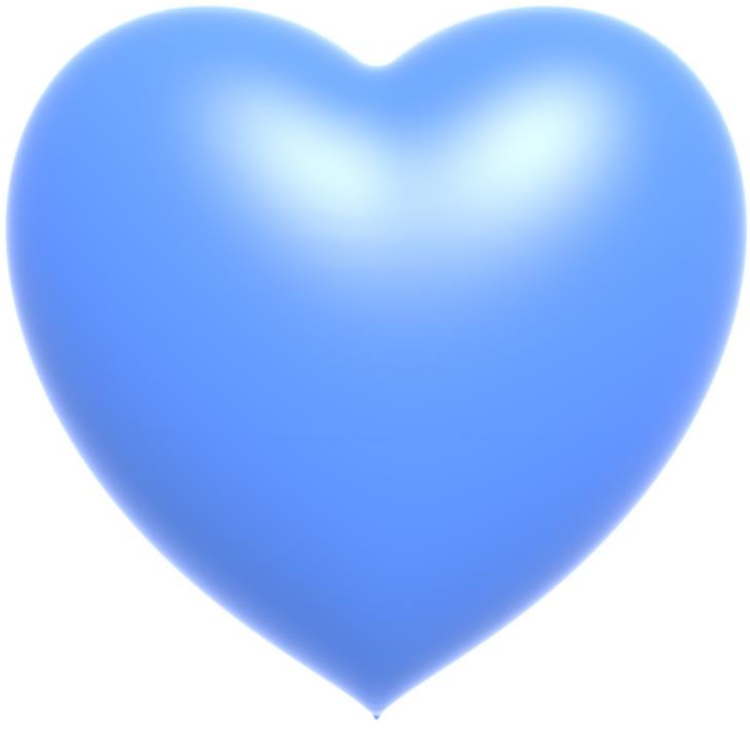
7/8 < 1

YES.



Implicit surfaces make some tasks easy (like inside/outside tests).

- **Here is another implicit surface**
  - **Can you give me Cartesian coordinates of a point that lies on it?**



$$(x^2 + \frac{9y^2}{4} + z^2 - 1)^3 = x^2 z^3 + \frac{9y^2 z^3}{80}$$

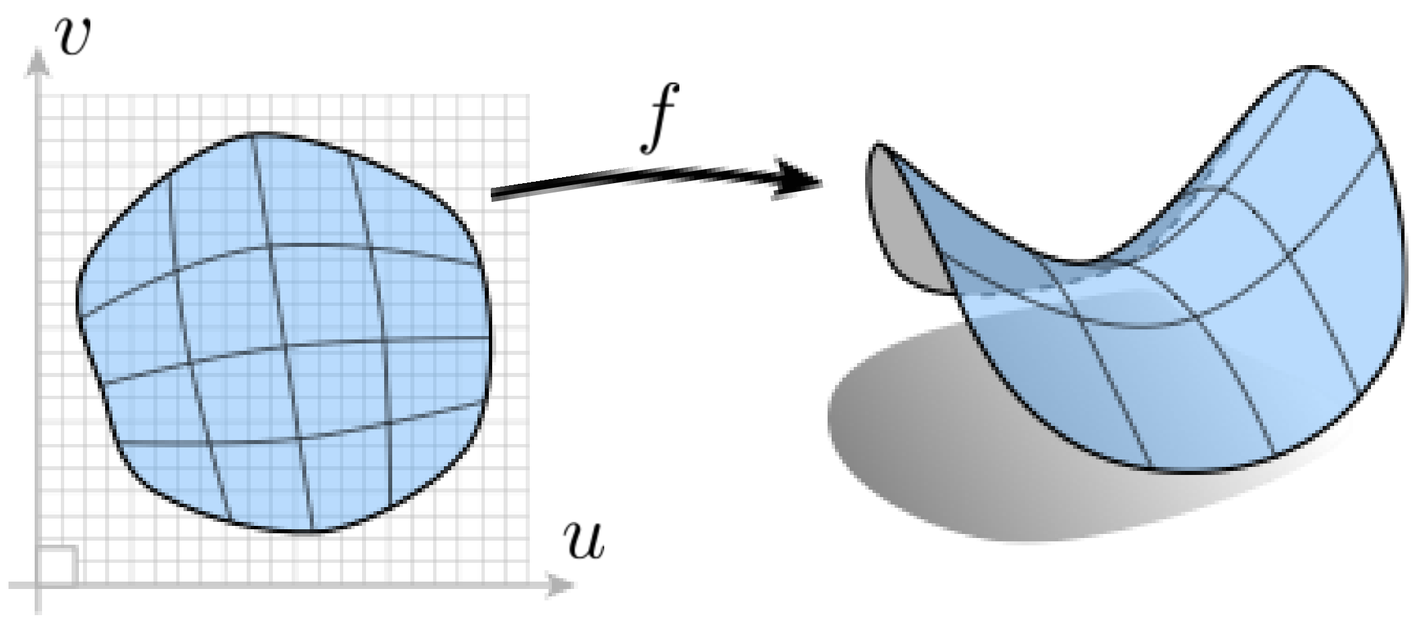
Implicit surfaces make some tasks hard (like sampling).

“Explicit” Representations of Geometry

- **All points are given directly**
- **E.g., points on sphere are**

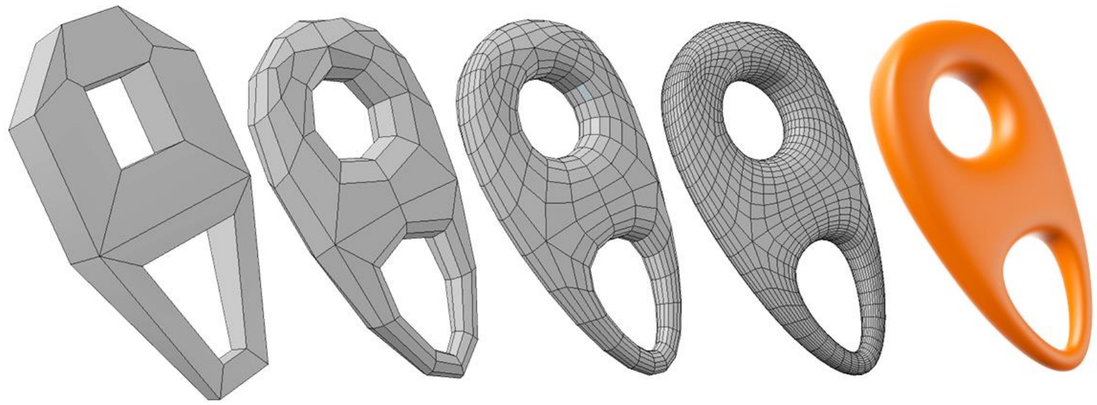
$$(\cos(u)\sin(v), \sin(u)\sin(v), \cos(v)),$$
$$\text{for } 0 \leq u < 2\pi \text{ and } 0 \leq v \leq \pi$$

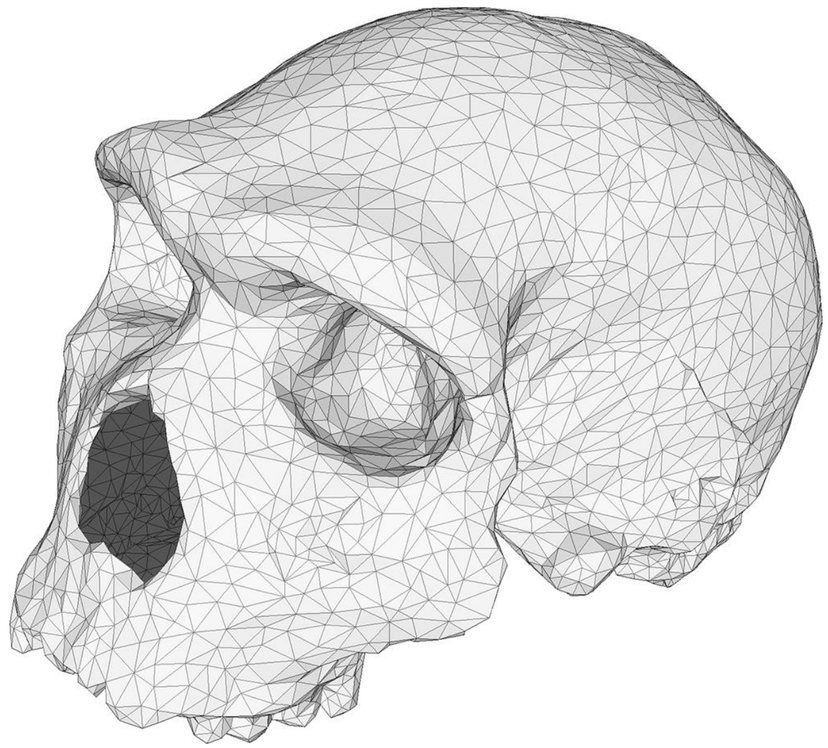
- **More generally:** $f : \mathbb{R}^2 \to \mathbb{R}^3; (u, v) \mapsto (x, y, z)$

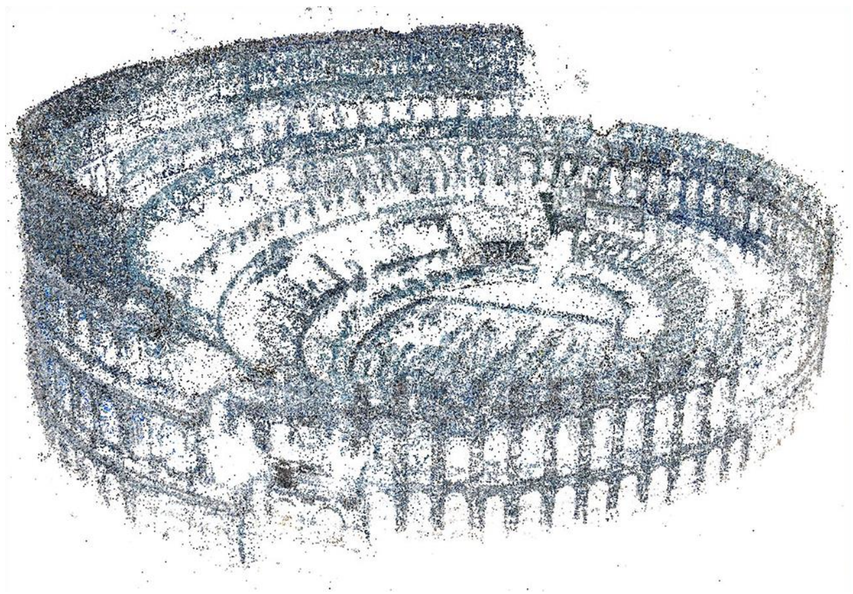


Many explicit representations in graphics

- **triangle meshes**
- **polygon meshes**
- **subdivision surfaces**
- **NURBS**
- **point clouds**
- **…**

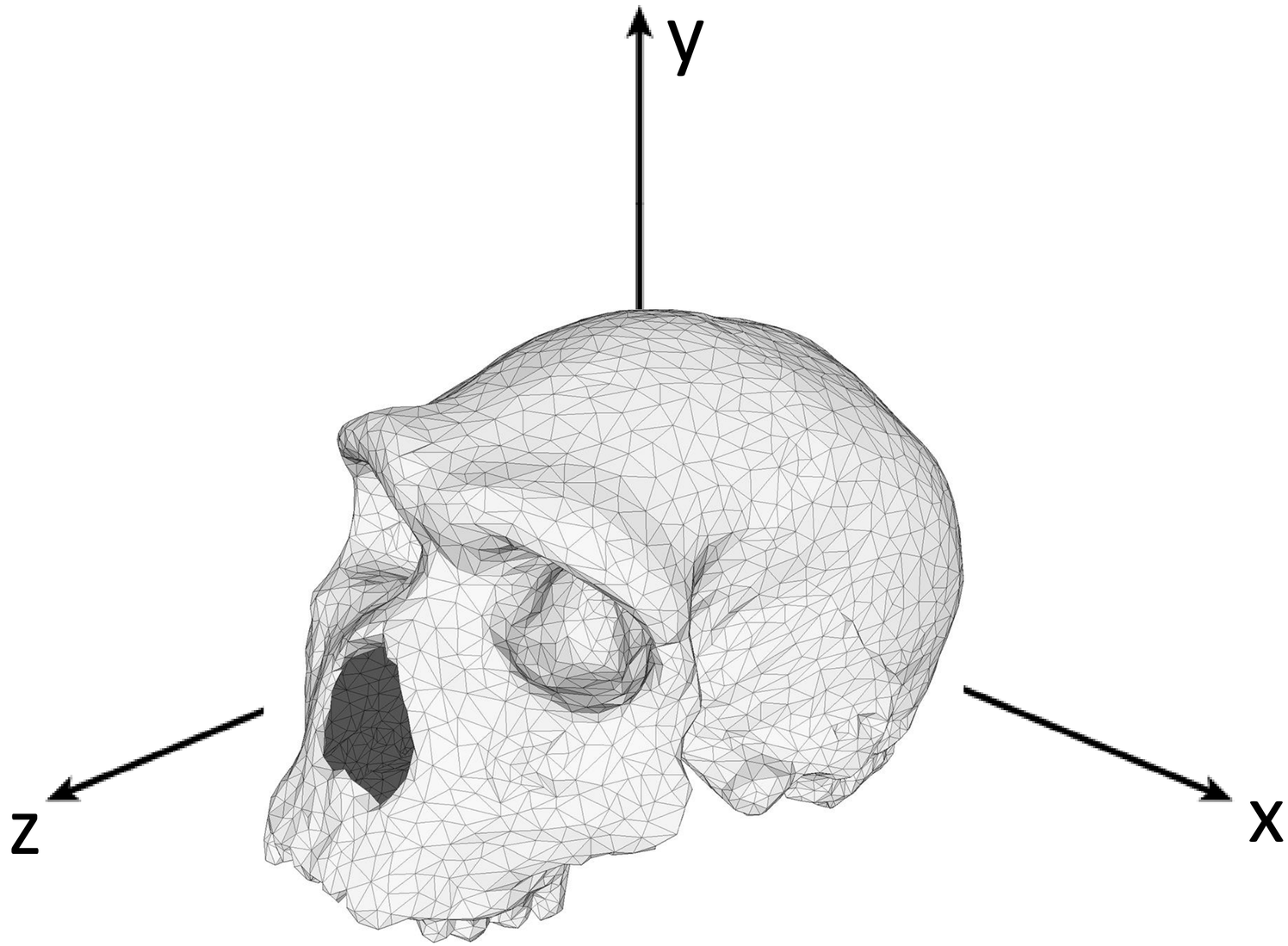






(Will see some of these a bit later.)

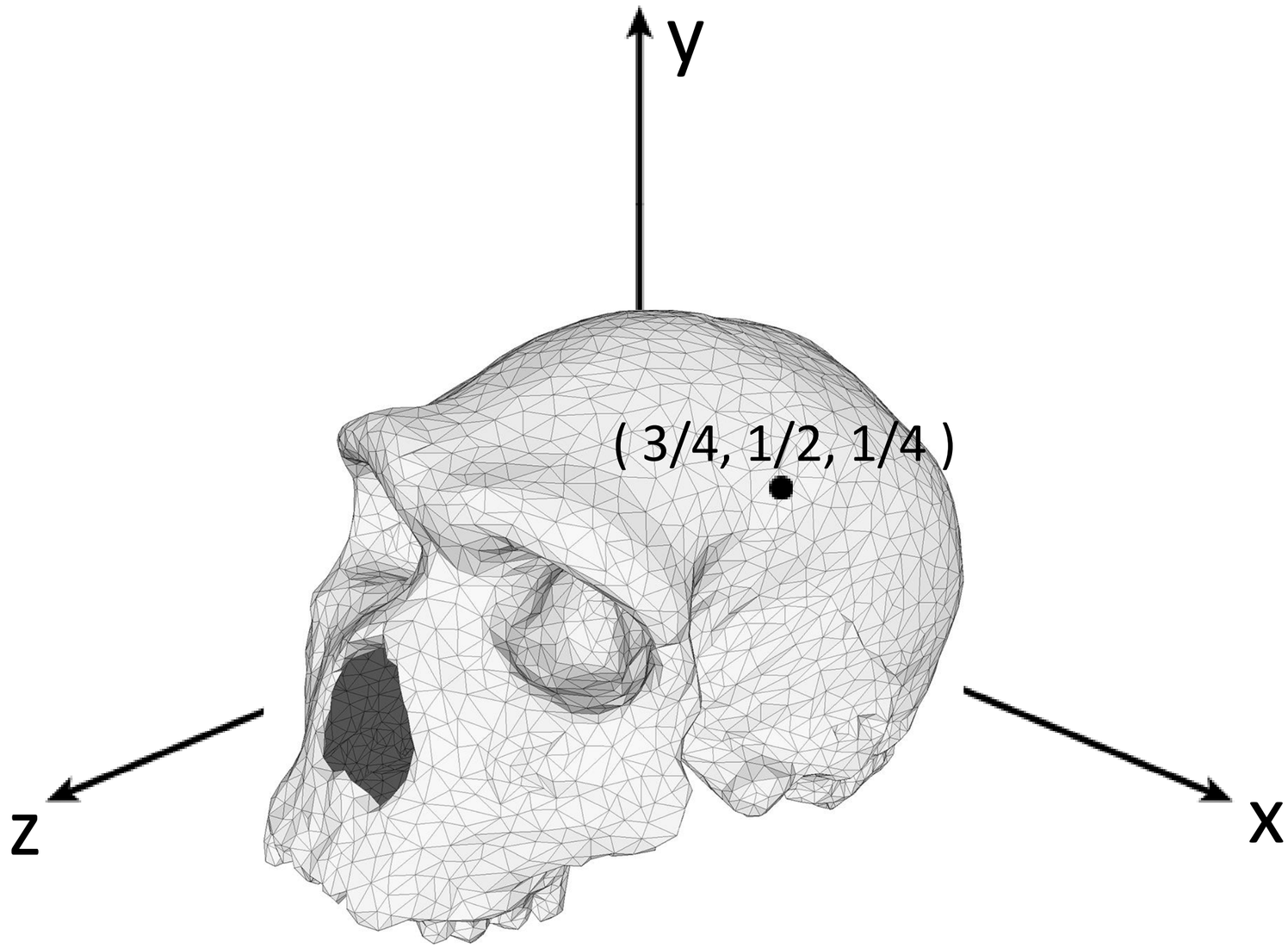
Sampling points on explicit surface



Explicit surfaces make some tasks easy (like sampling).

Check if this point is inside the skull

How about the point (3/4, 1/2, 1/4)?



Explicit surfaces make some tasks hard (like inside/outside tests).

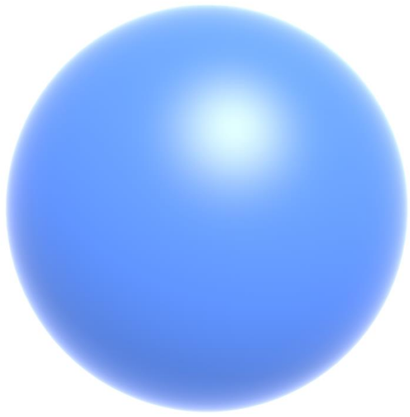
Different representations are better suited for different types of geometry and different types of operations we may want to perform.

Let’s take a closer look at some common representations used in computer graphics.

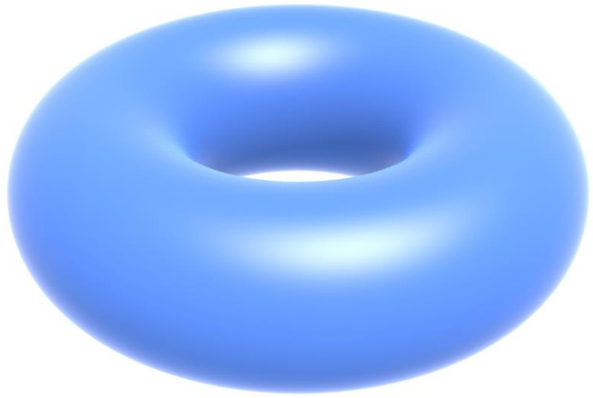
Algebraic Surfaces (Implicit)

Surface is zero set of a polynomial in x, y, z ("algebraic variety")

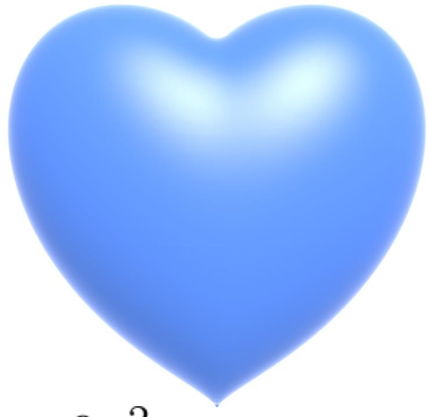
Examples:



$x^2+y^2+z^2=1$



$(R-\sqrt{x^2+y^2})^2+z^2=r^2$



$(x^2+\frac{9y^2}{4}+z^2-1)^3=$

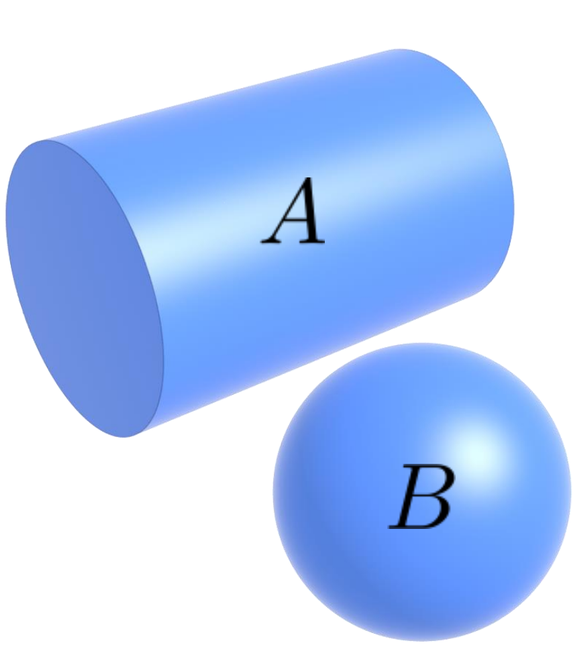
$$x^2z^3+\frac{9y^2z^3}{80}$$

What about more complicated shapes?

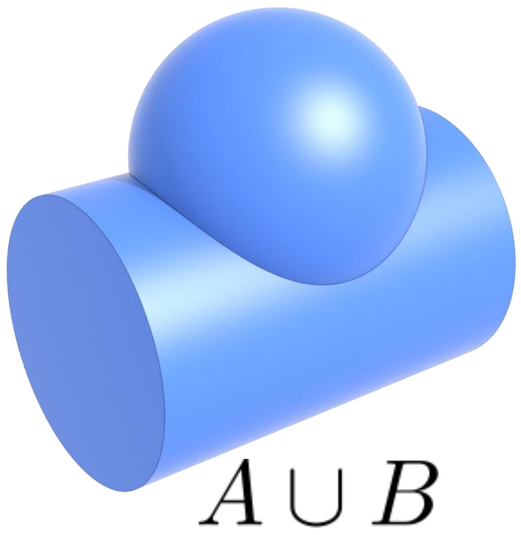
Very hard to come up with polynomials!

Constructive Solid Geometry (Implicit)

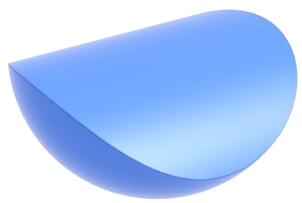
Build complicated shapes via Boolean operations



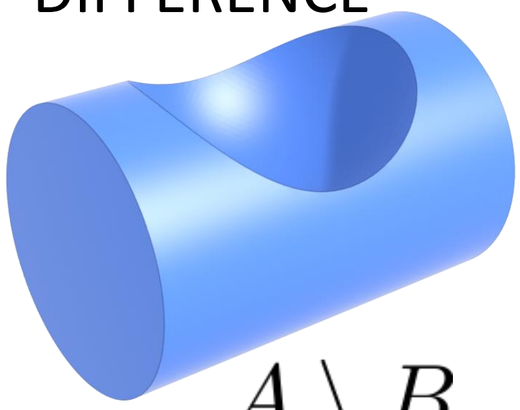
UNION



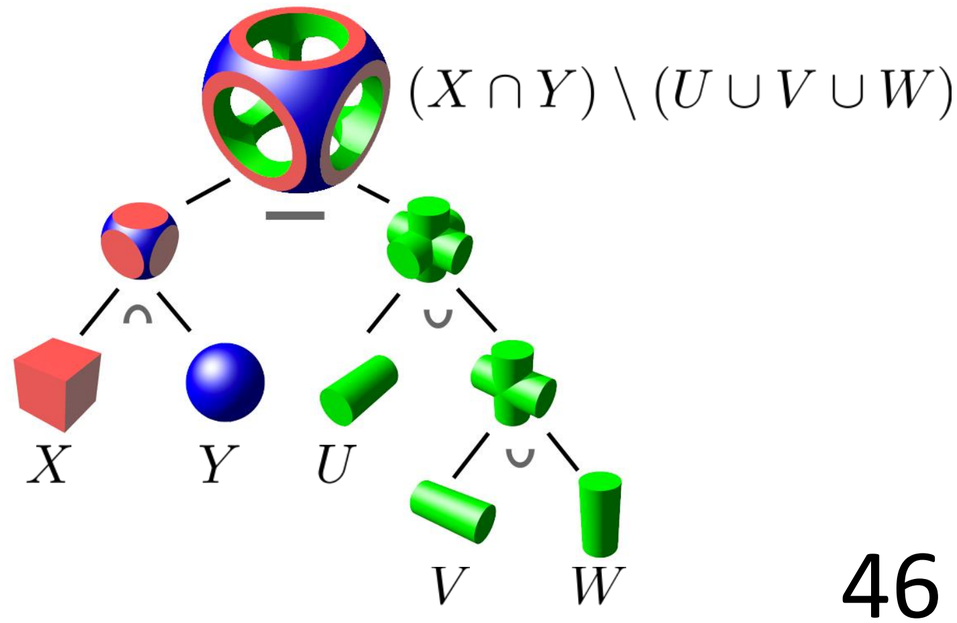
INTERSECTION



DIFFERENCE

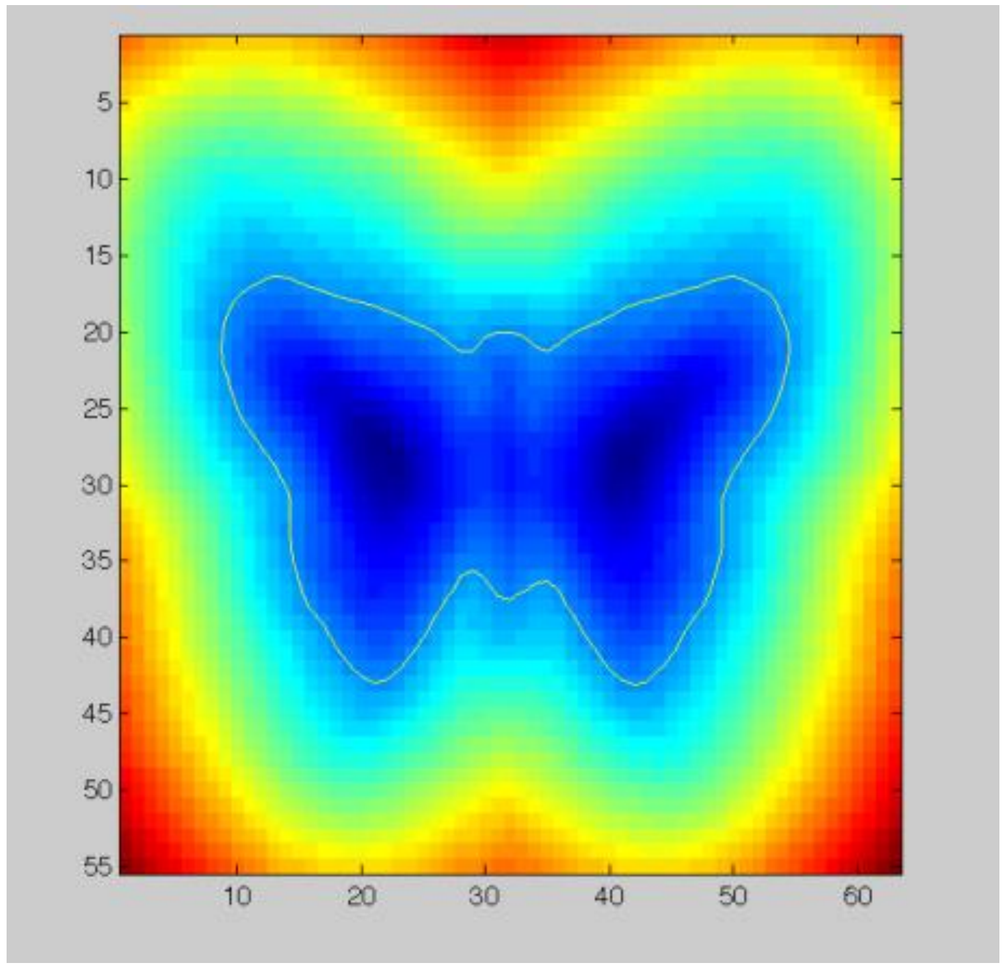


Then chain together:



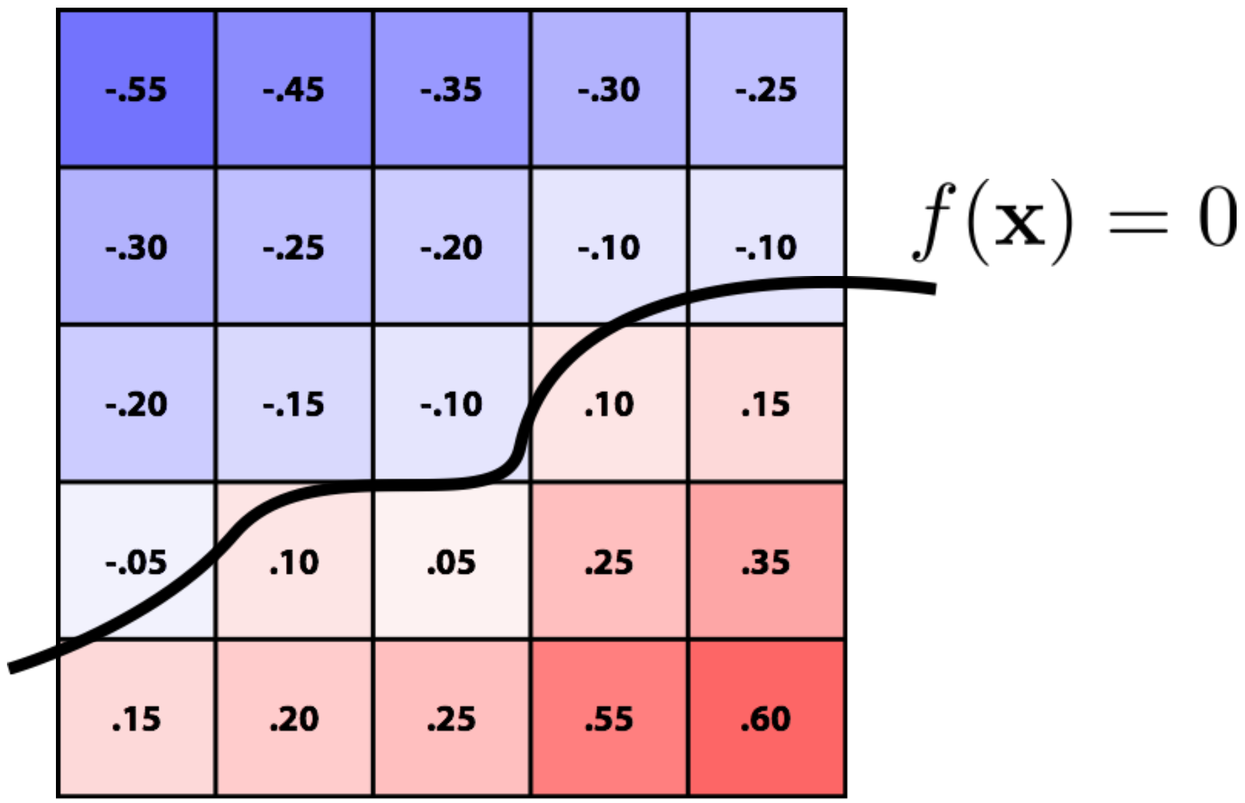
Distance Functions (Implicit)

- **A *distance function* gives distance to closest point on object**



Level Set Methods (Implicit)

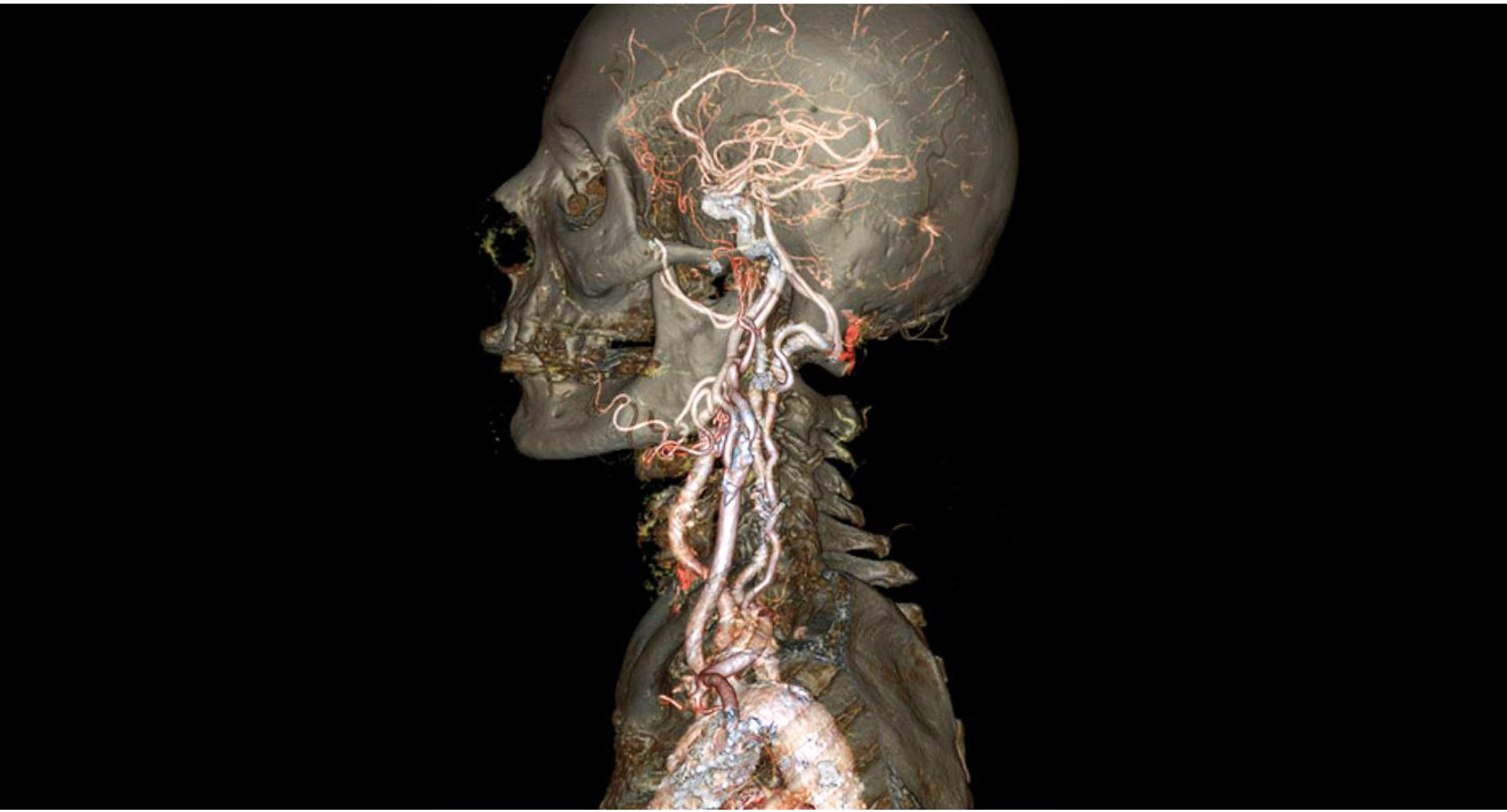
- Implicit surfaces have some nice features (e.g., merging/splitting)
- But, hard to describe complex shapes in closed form
- Alternative: store a grid of values approximating function



- Surface is found where *interpolated* values equal zero
- Provides much more explicit control over shape

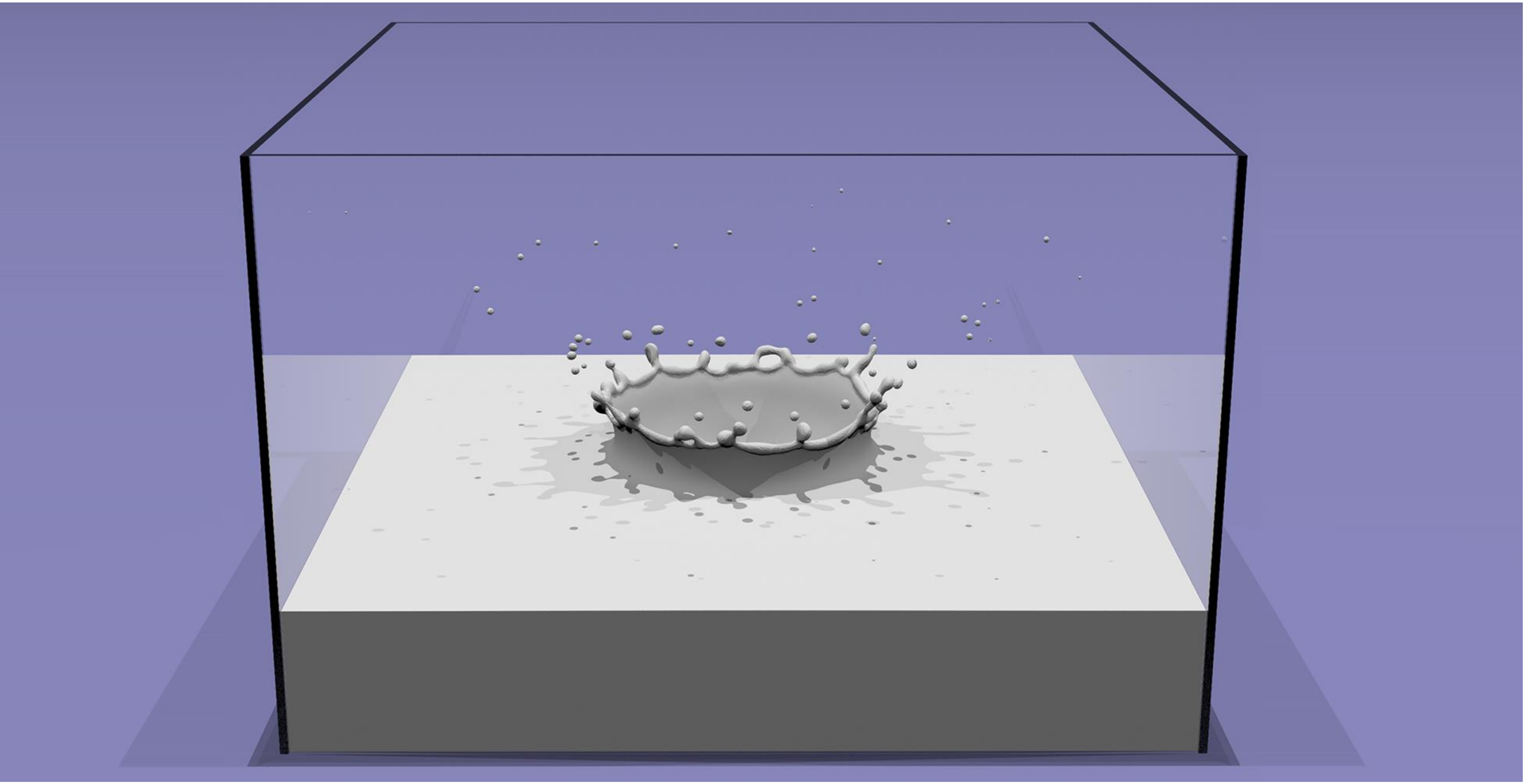
Level Sets from Medical Data (CT, MRI, etc.)

Level sets encode, e.g., constant tissue density



Level Sets in Physical Simulation

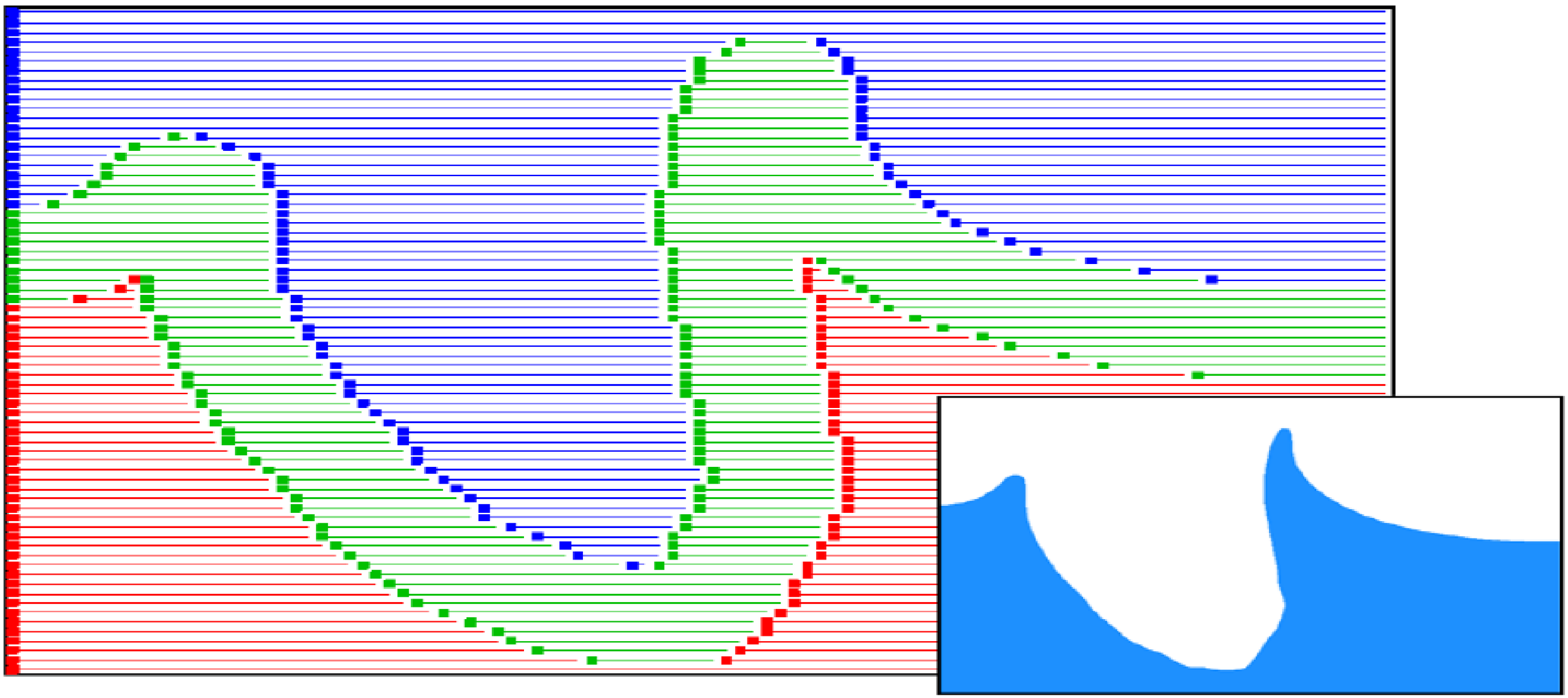
Level set encodes distance to air-liquid boundary



See http://physbam.stanford.edu

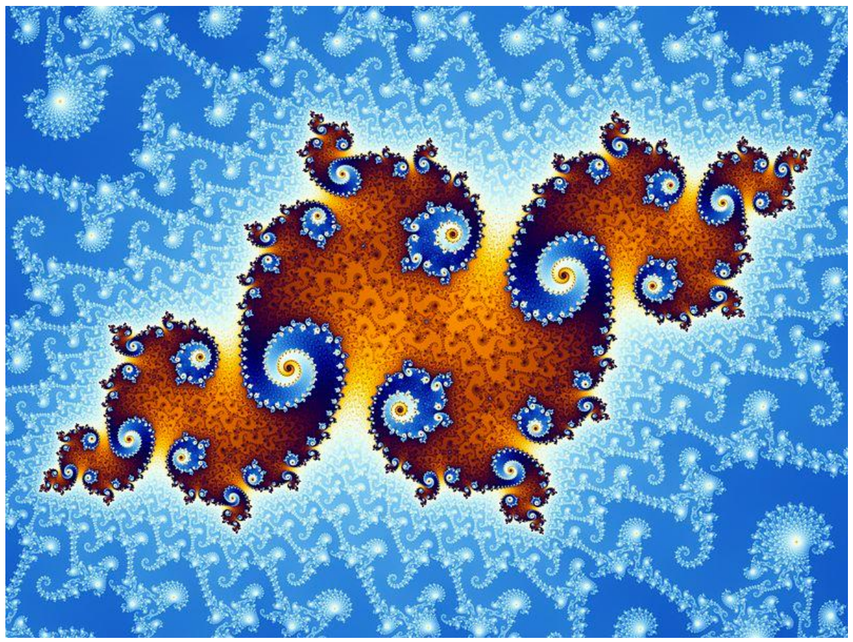
Level Set Storage

- **Drawback: storage for 2D surface is now $O(n^3)$**



Fractals and L-systems (Implicit)

- No precise definition; structures that exhibit self-similarity, detail at all scales
- New “language” for describing natural phenomena
- Hard to control shape!



L-Systems (Implicit)

Underlying computational system is `simple’

- geometry governed by simple rules
- the end result has interesting, useful, and/or surprising properties
- L = Lindenmayer, the inventor of these types of systems

L-Systems

- An L system is a set of grammar rules,
- Plus a *start symbol* (or an initial group of symbols)
- Plus *semantics* – a way of interpreting the grammar strings

E.g. here is a grammar:

rule 1: A → BBC

rule 2: B → abA

Here is a start string:

AAA

Here's what happens when we apply rules a few times:

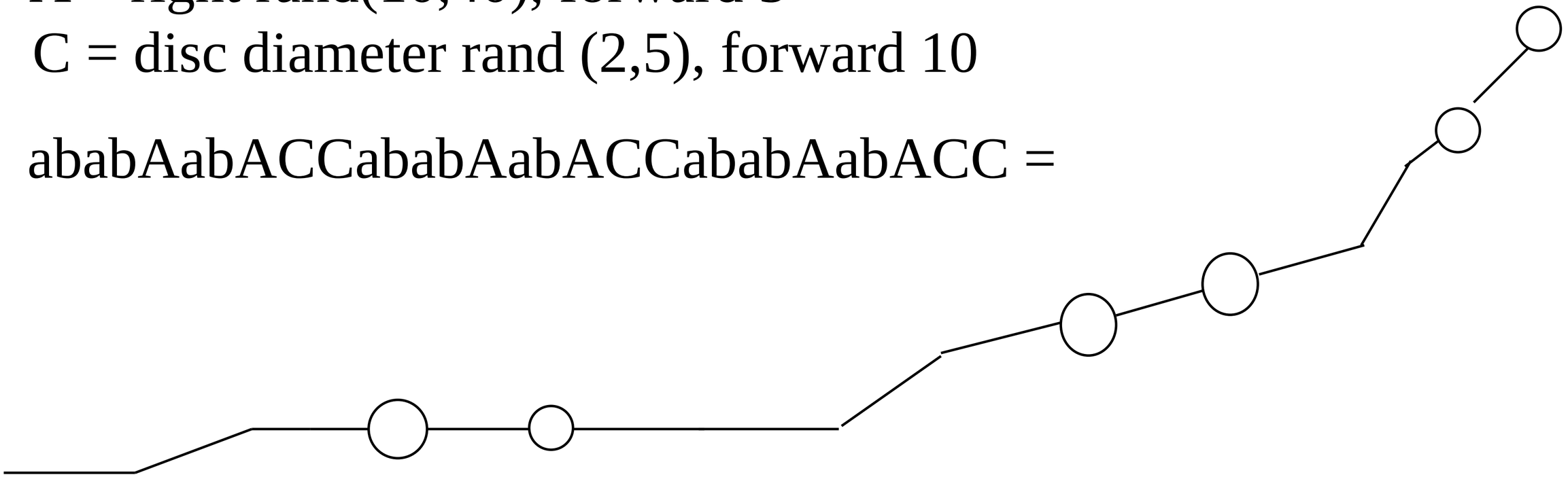
AAA → BBCBBCBBC → abACabACabAC →
abBBCCabBBCCabBBCC → ababAabACCababAabACCababAabACC
Etc …

L-Systems

AAA → BBCBBCBBC → abACabACabAC →
abBBCCabBBCCabBBCC → ababAabACCababAabACCababAabACC

Things become interesting when we add the *semantics*. In this case,
Let: a = forward 10
b = left rand(10,40)
A = right rand(10,40), forward 5
C = disc diameter rand (2,5), forward 10

Then ababAabACCababAabACCababAabACC =



A more interesting examples

- **Grammar symbols: {F,+,–}**
  - **Rules: F→F+F--F+F**
  - **Start symbol (or, *axiom*): F**
- **Note that when a rule is applied in an L system, it is applied simultaneously to all possible positions.**
- **In this case, after just 2 applications of the rule we get:**

F+F--F+F + F+F--F+F -- F+F--F+F + F+F--F+F

Now let:

- **“F” – Draw forward one unit**
- **“+” – Rotate left**
- **“–” – Rotate right**

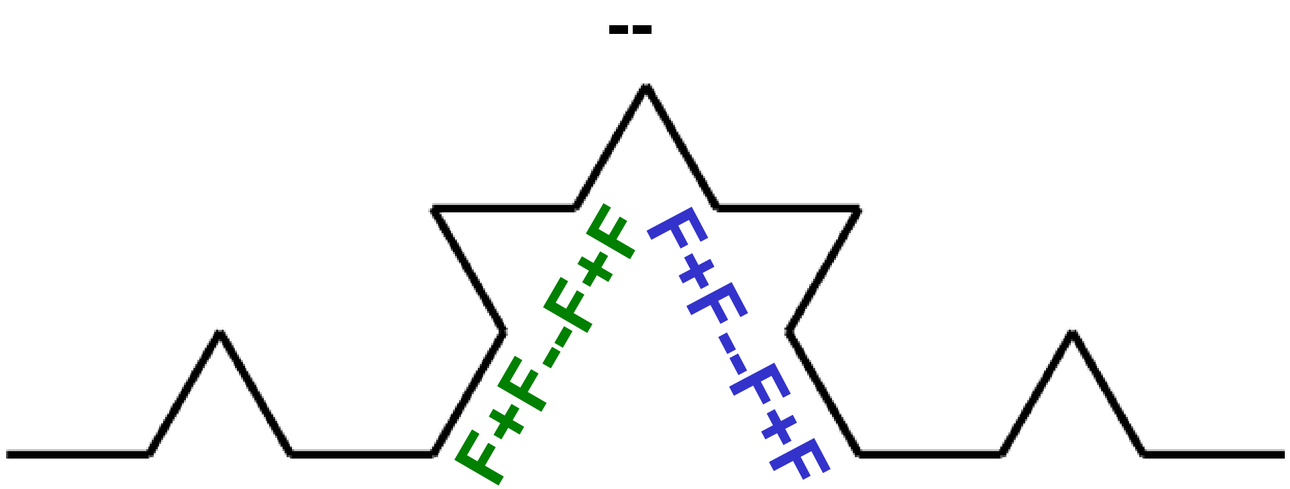
- **Units and angles are global constants**
- **Commands change “cursor” state (position, orientation)**

von Koch Snowflake Curve

+,- = 60º

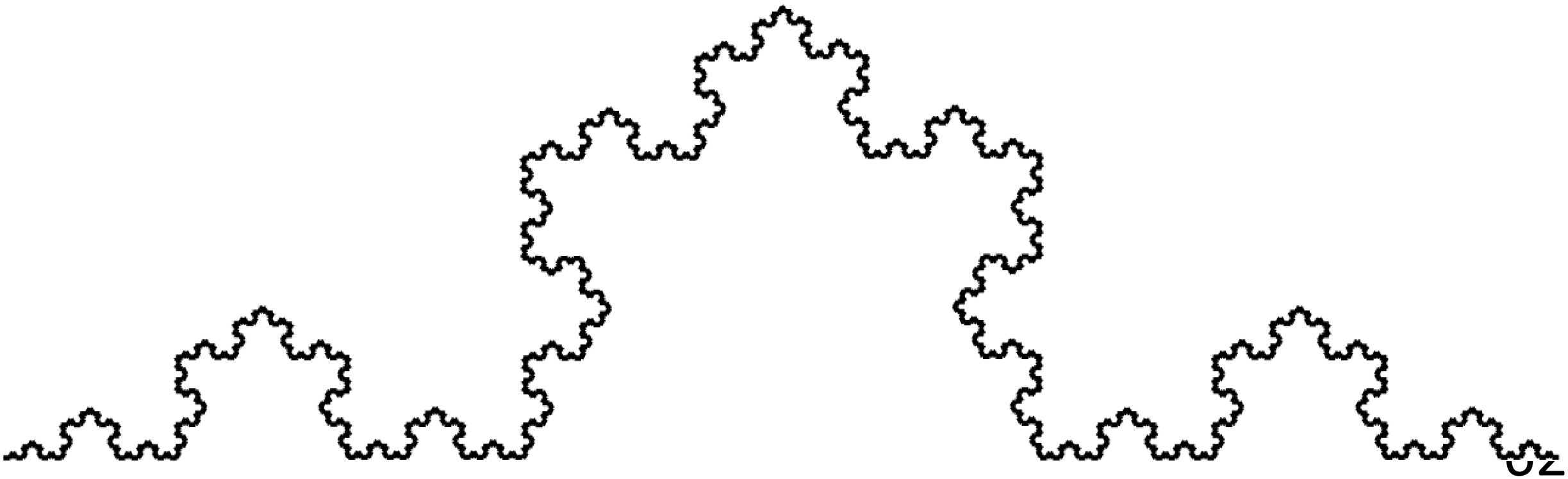
F

F+F--F+F



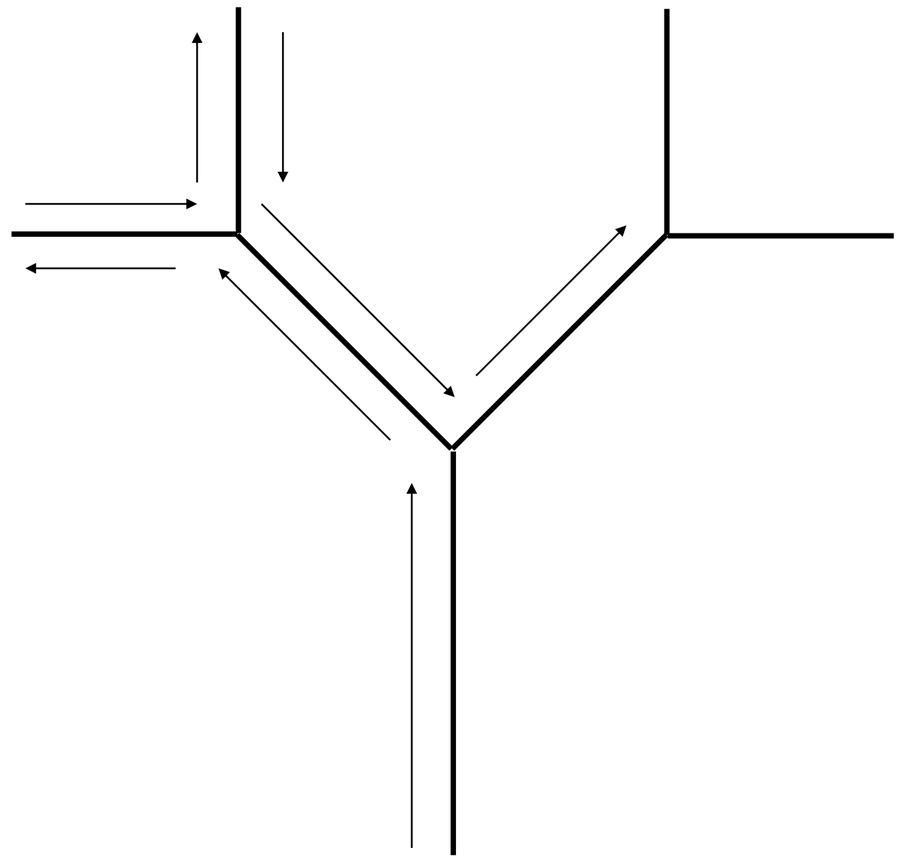
F+F--F+F +

+ F+F--F+F



Branching structures

- **Assume:**
- **- “[” – pushes state onto stack**
  - **“]” – pops state from stack**
- **Use “[” to start a branch and “]” when finished to return to its base**
- **So include “[” and “]” in the grammar rule(s).**



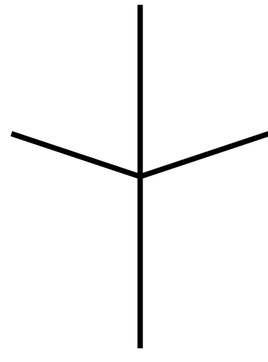
F+F+F----++F----+…

vs.

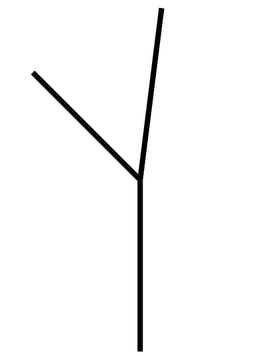
F[+F[+F][-F]][-F[+F][-F]]

Rules for Types of Branching

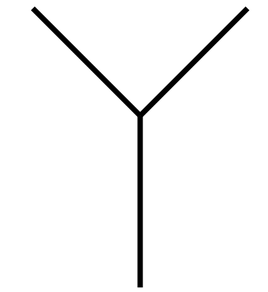
- **Monopodial**
  - **Trunk extends undeviated to top**
  - **Branches extend perpendicularly** T→T[+B][-B]T



- **Sympodial**
  - **Trunk deviates to top**
  - **Branches extend perpendicularly** T→T[---B]+T

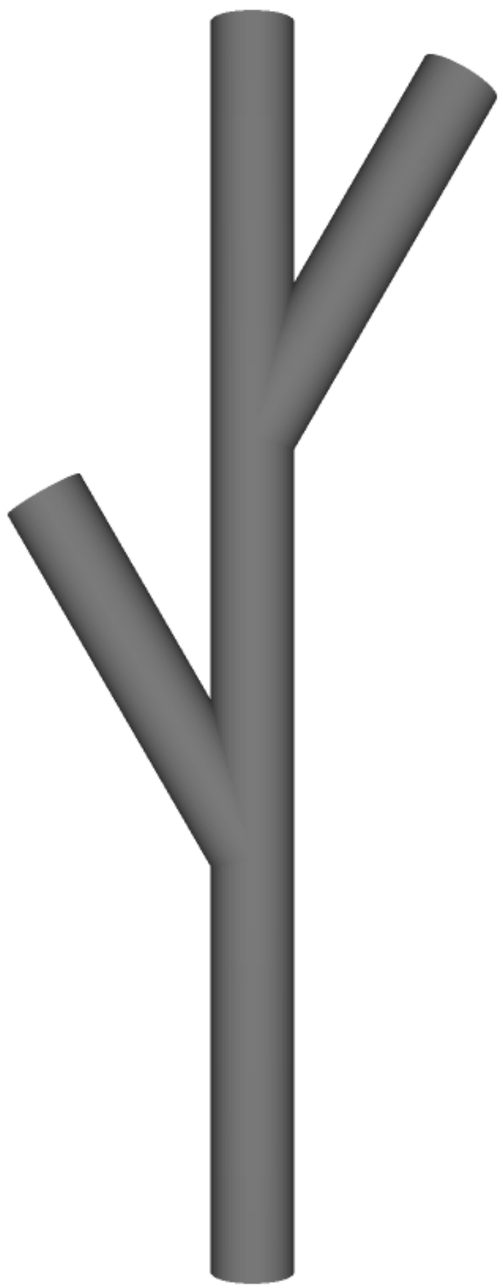


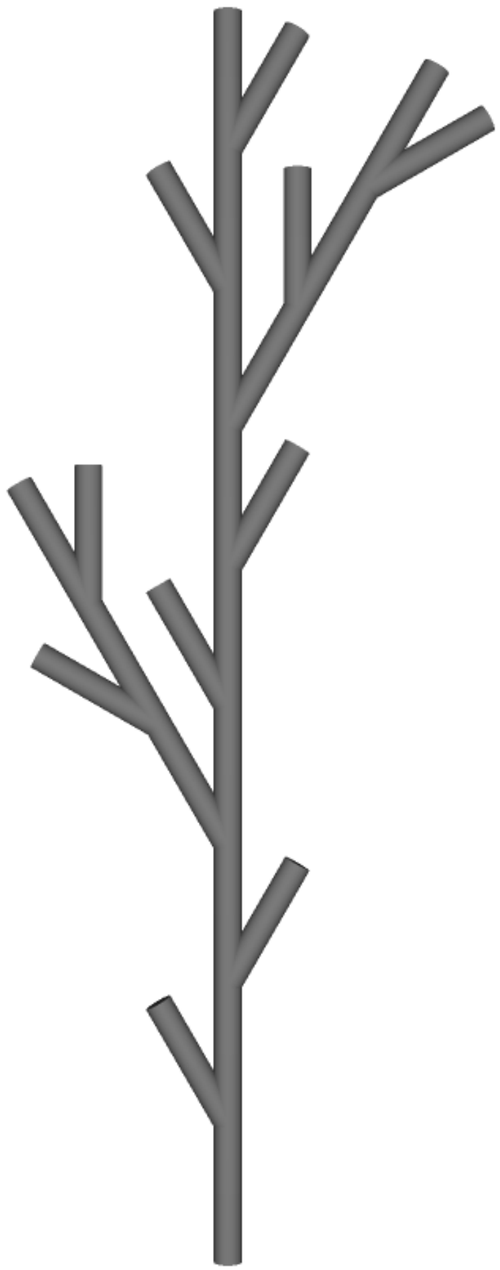
- **Binary**
  - **Trunk terminates at first branching point**
  - **Branches deviate uniformly** T→T[+B][-B]

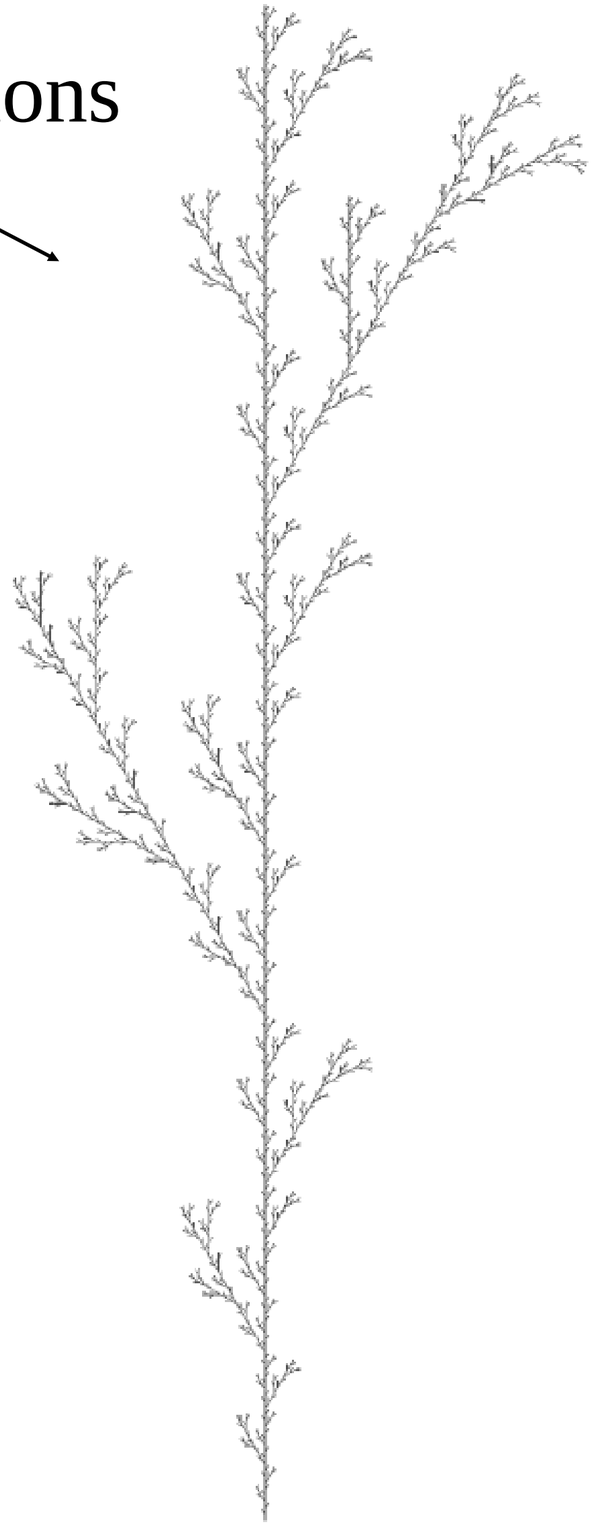


Example of this L system: F→F[+F]F[-F]F

Axiom = F. 1 application; 2 applications; 4 applications







Stochastic L-Systems

- **Conditional firing of rules**

  F → <something>: probability1

  F → <something2>: probability2

  etc …

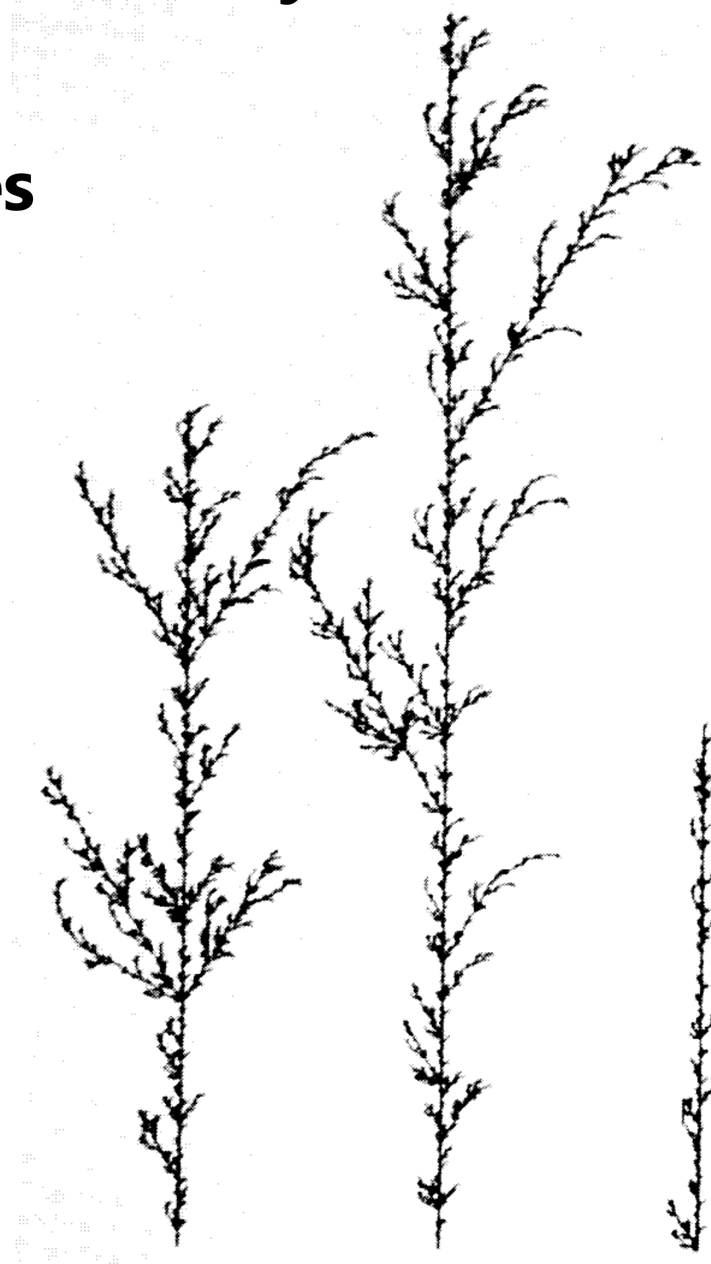
- **Example: Random Bushes**

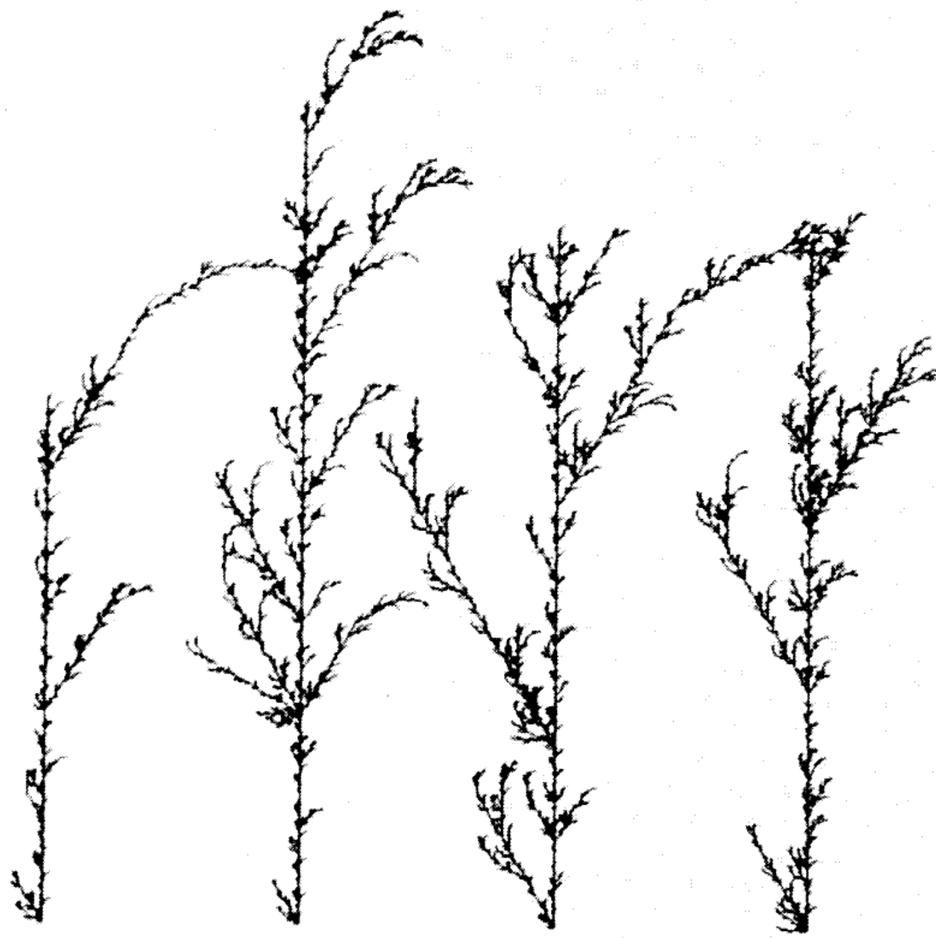
  F → F[+F]F[-F]F : 0.33

  F → F[+F]F : 0.33

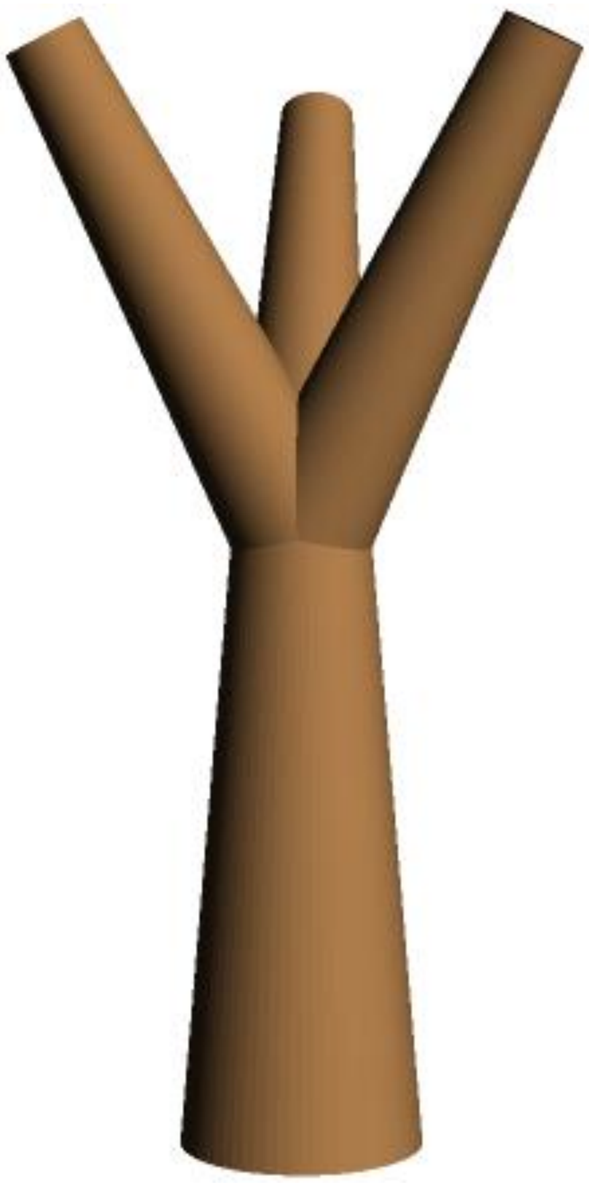
  F → F[-F]F : 0.34

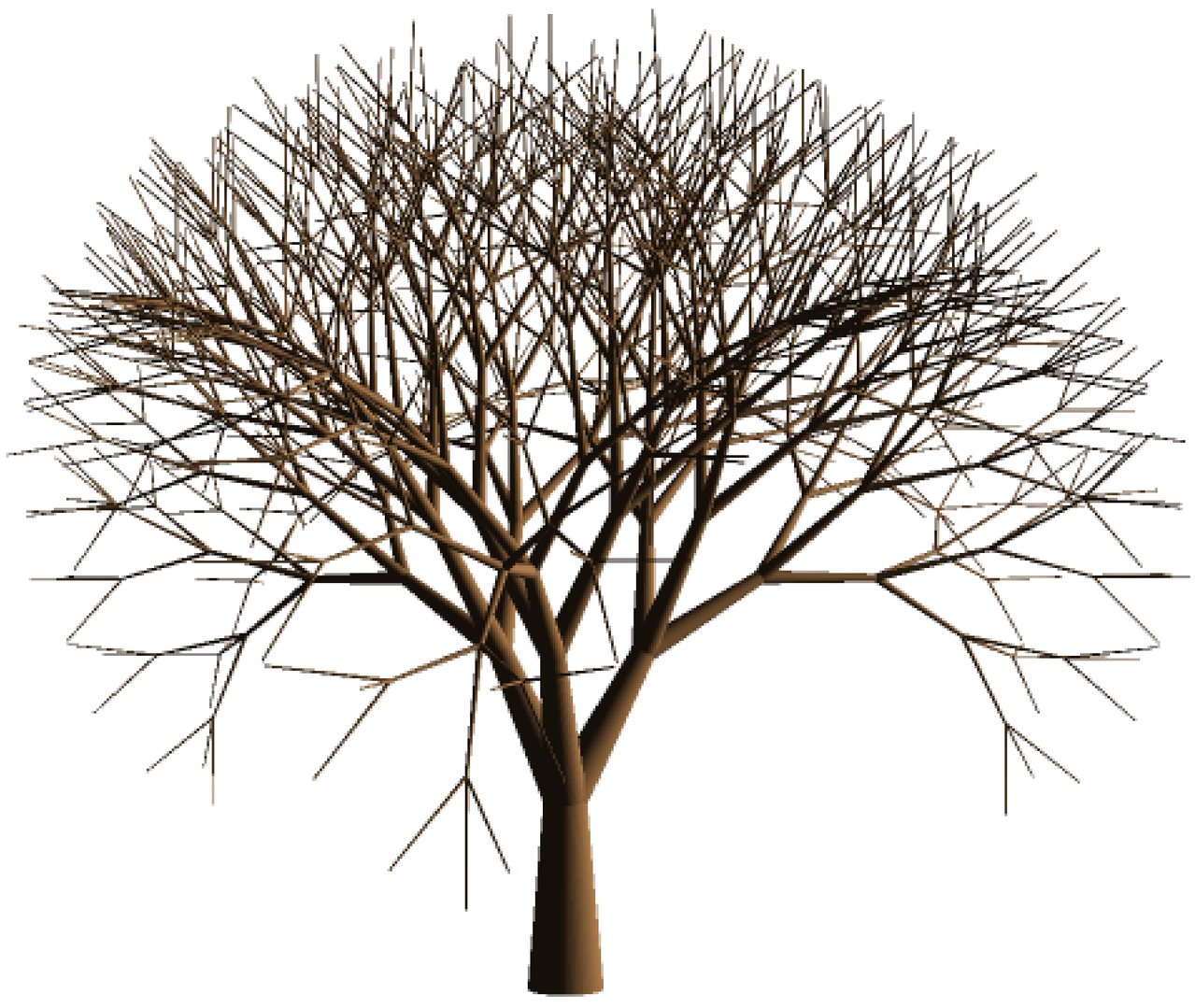
- **If not stochastic, then *deterministic***



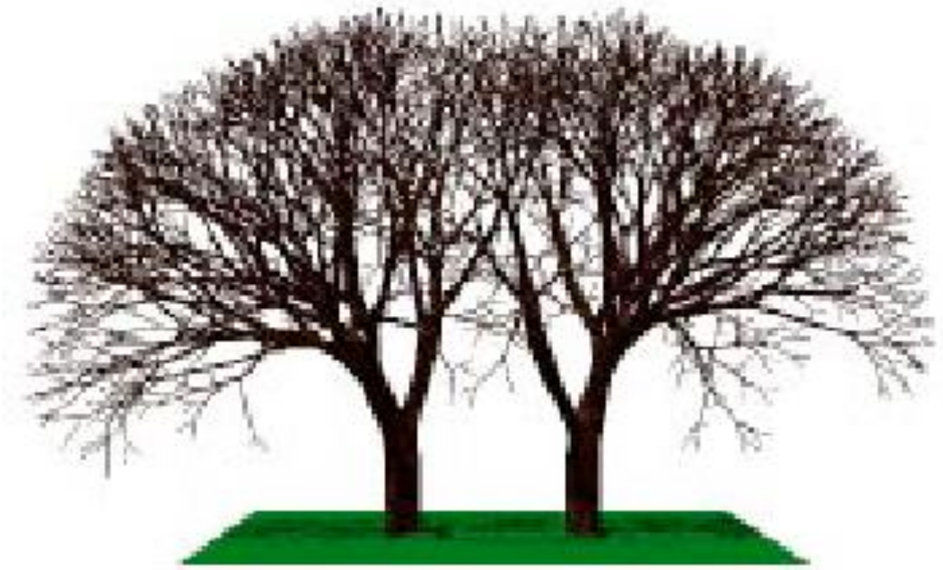


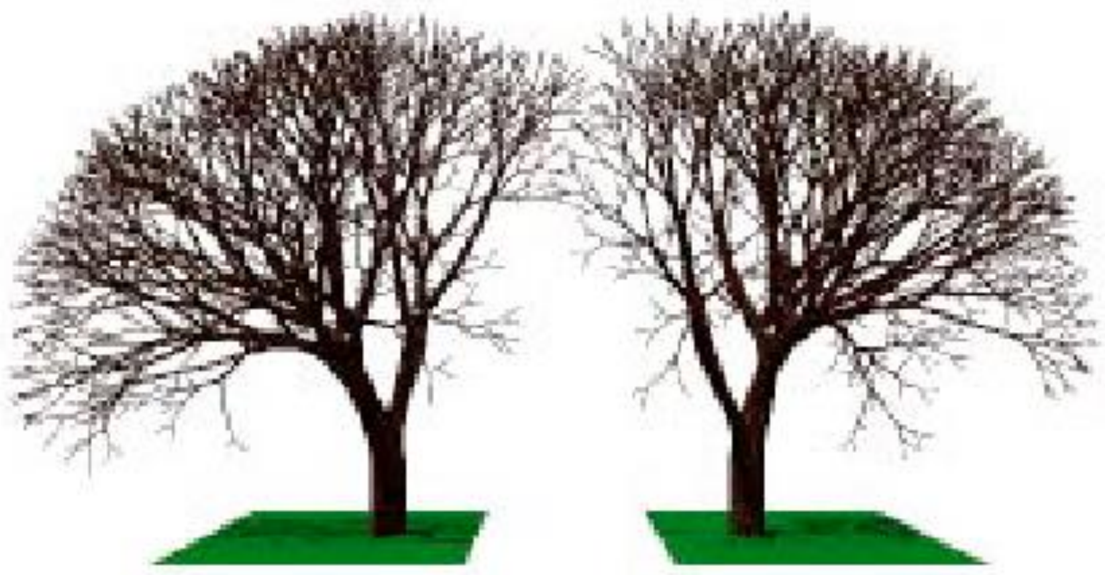
Ternary Tree, using only: F → F[&F][&/F][&\F]





Can get pretty complex





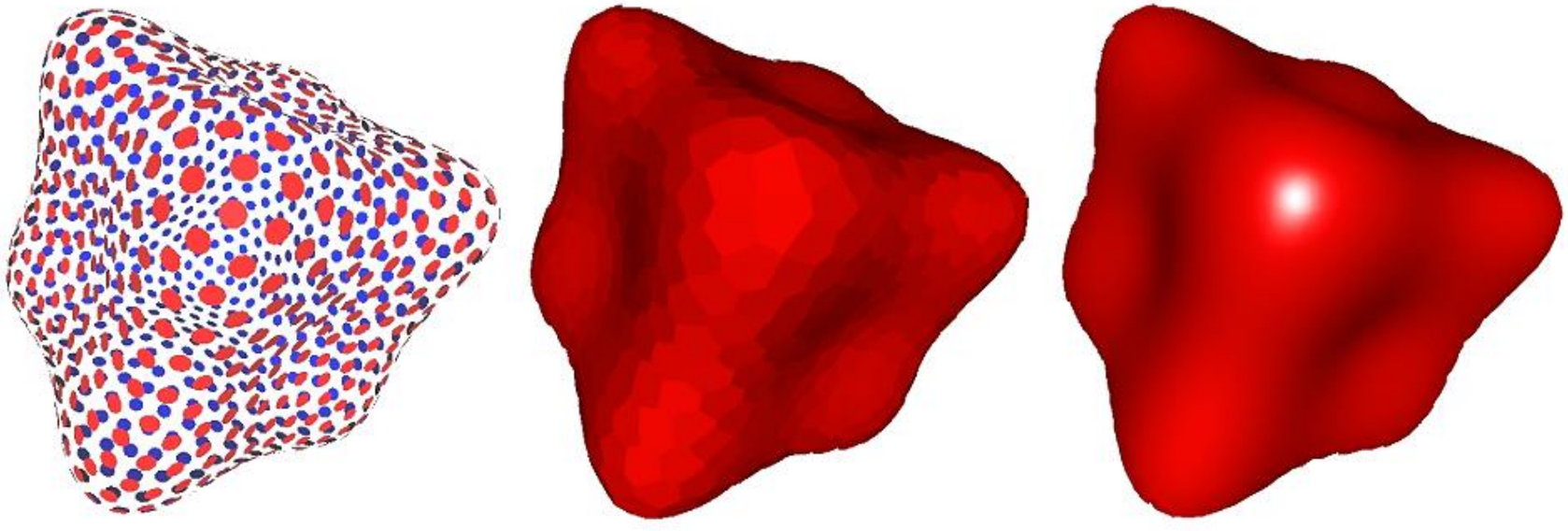
Implicit Representations - Pros & Cons

- **Pros:**
  - **description can be very compact (e.g., a polynomial)**
  - **Can be easy to determine if a point is inside/outside (just plug it in!)**
  - **other queries may also be easy (e.g., distance to surface)**
  - **for simple shapes, exact description/no sampling error**
  - **easy to handle changes in topology (e.g., fluid)**
- **Cons:**
  - **expensive to find all points in the shape (e.g., for drawing)**
  - ***very difficult to model complex shapes***

What about explicit representations?

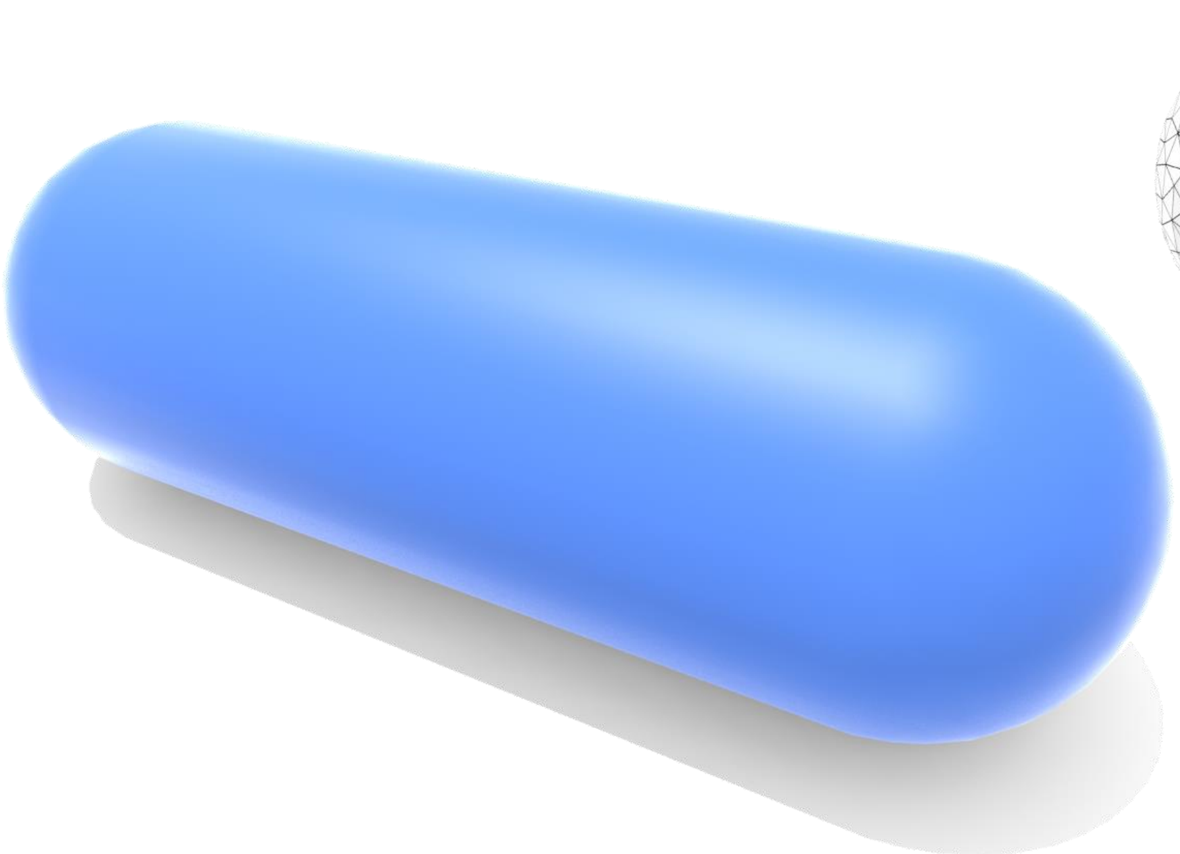
Point Cloud (Explicit)

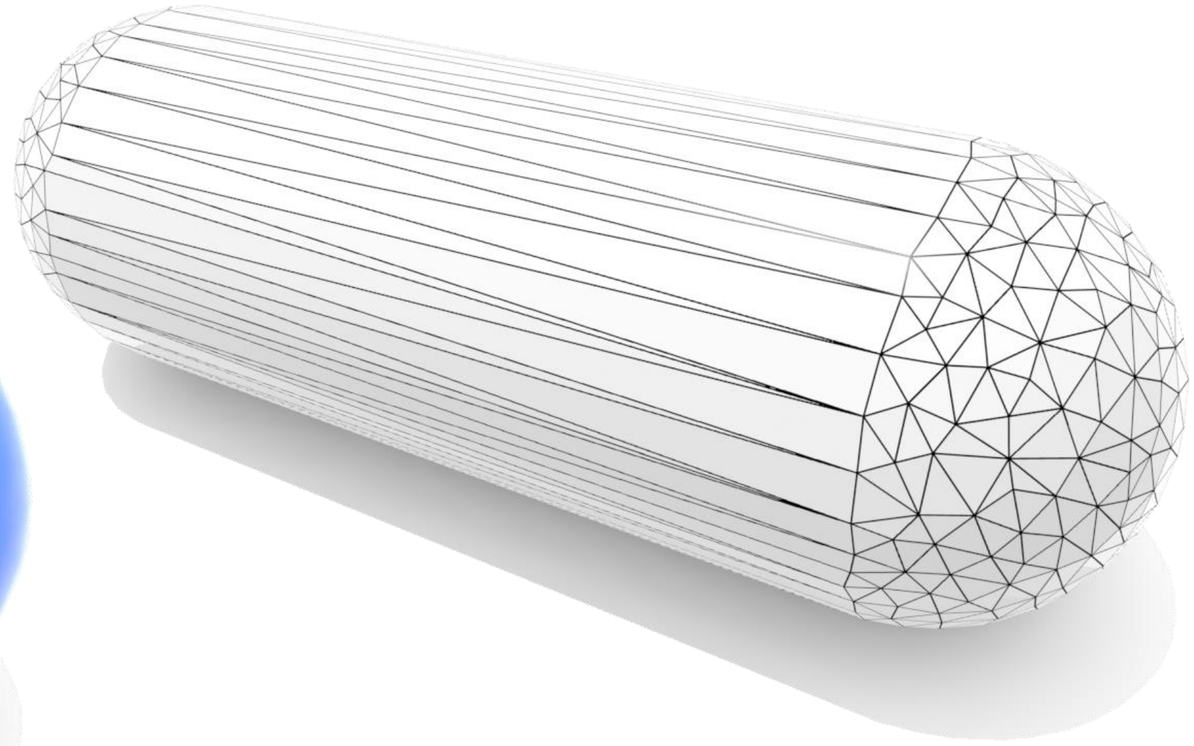
- **Simplest representation: list of points (x,y,z)**
- **Often augmented with *normals***
- **Easily represent any kind of geometry**
- **Useful for LARGE datasets (>>1 point/pixel)**
- **Difficult to draw in undersampled regions**
- **Hard to do processing / simulation**



Polygonal Mesh (Explicit)

- **Store vertices *and* polygons (most often triangles or quads)**
- **Easier to do processing/simulation, adaptive sampling**
- **More complicated data structures**
- **Perhaps most common representation in graphics**

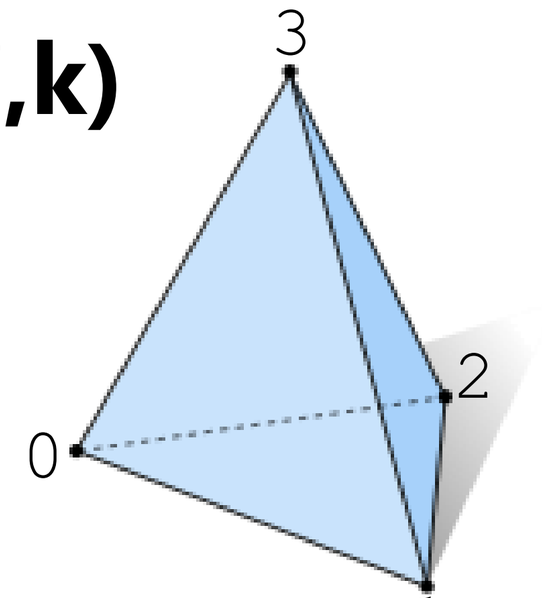




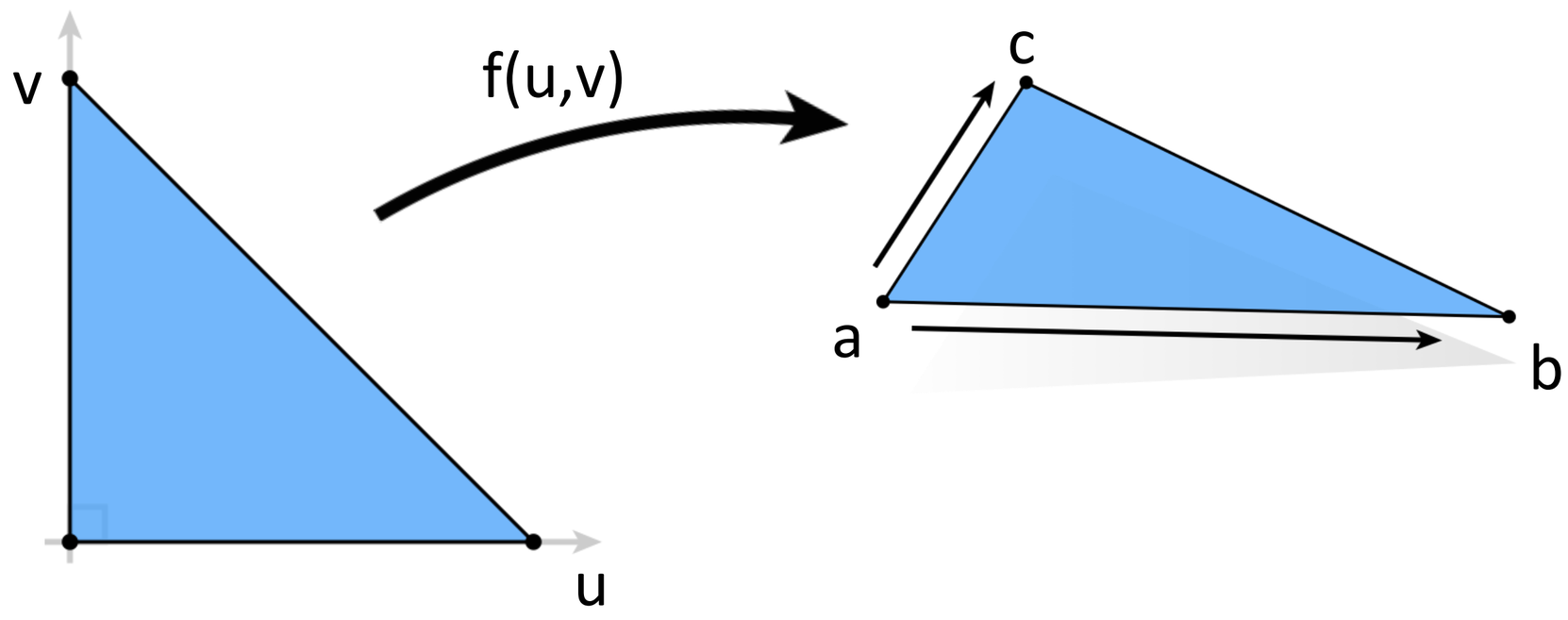
Triangle Mesh (Explicit)

- **Store vertices as triples of coordinates (x,y,z)**
- **Store triangles as triples of indices (i,j,k)**
- **E.g., tetrahedron:** VERTICES TRIANGLES

	x	y	z		i	j	k
0:	-1	-1	-1		0	2	1
1:	1	-1	1		0	3	2
2:	1	1	-1		3	0	1
3:	-1	1	1		3	1	2



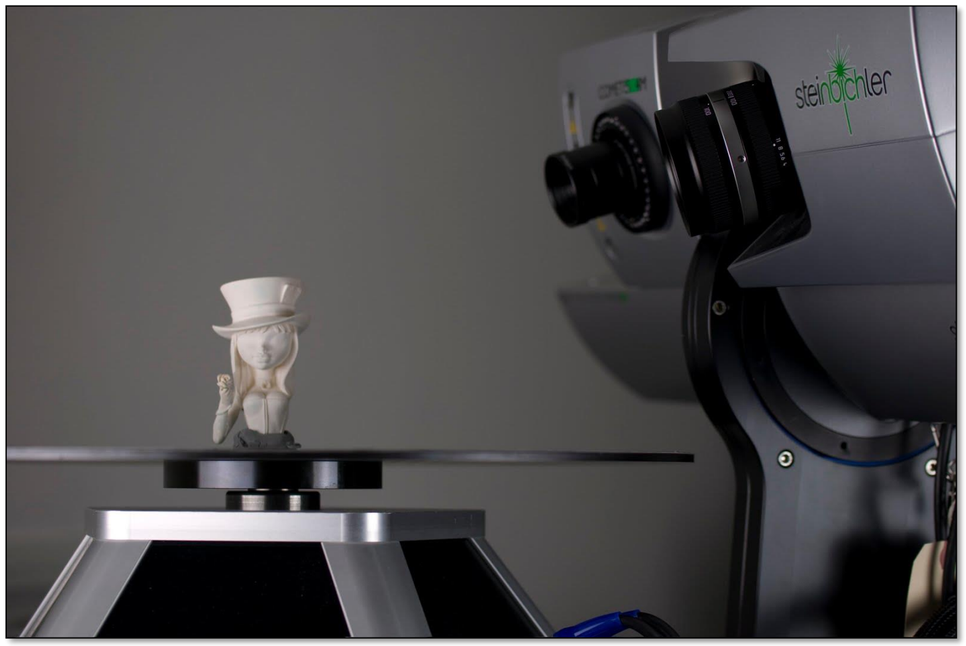
- Can think of triangle as *affine* map from plane into space:



But where is this geometry coming from?

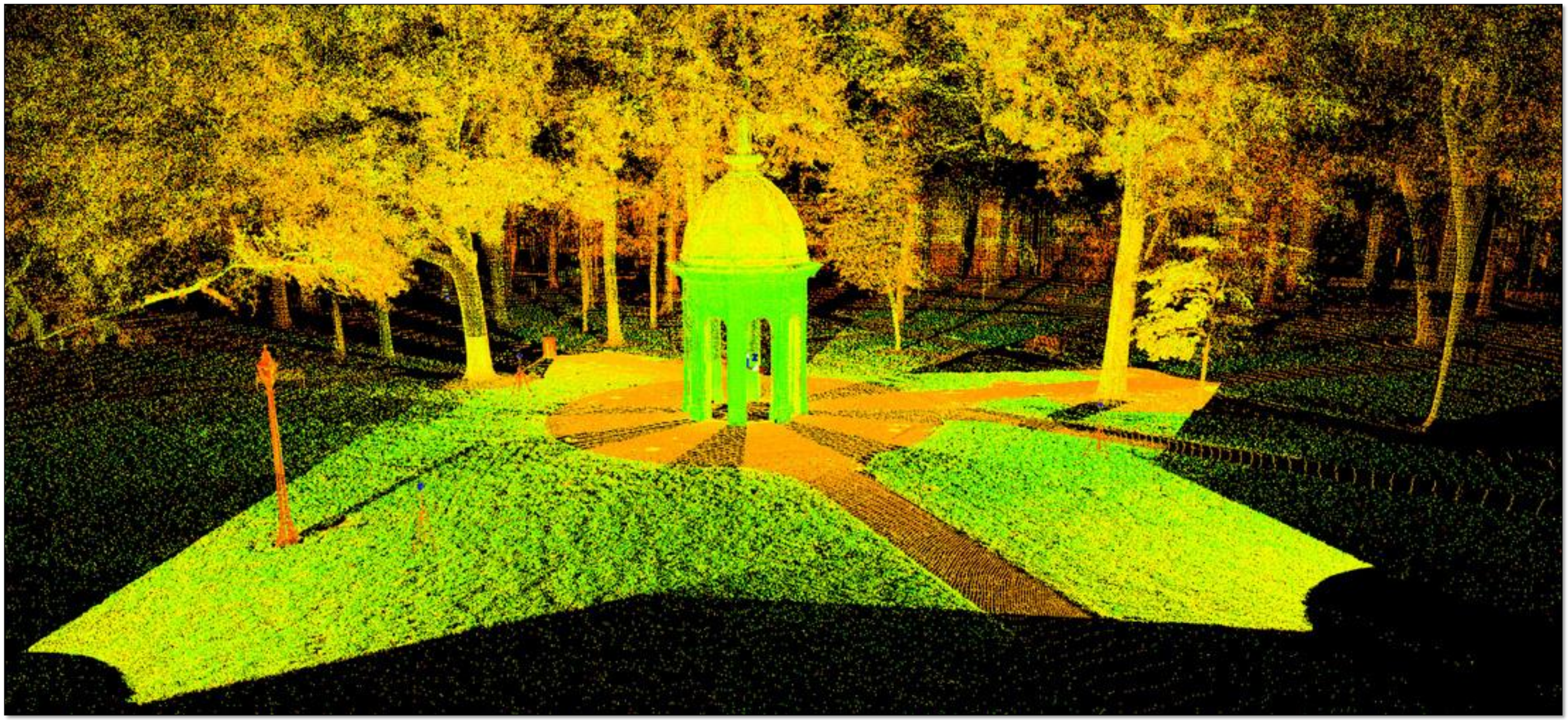
Sources of geometry

- **Acquired real-world objects via 3D Scanning**



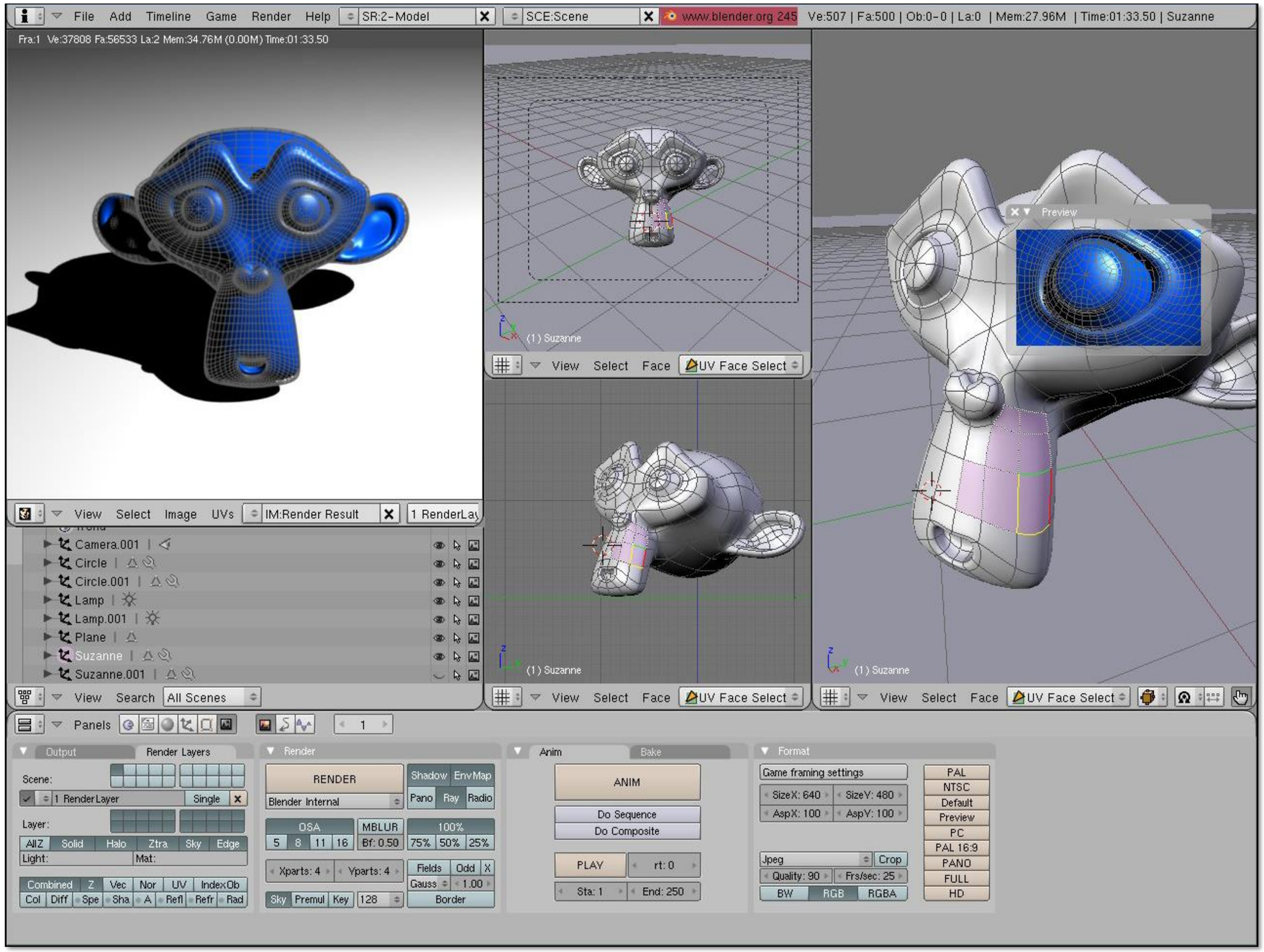
Sources of geometry

- **Acquired real-world objects via 3D Scanning**



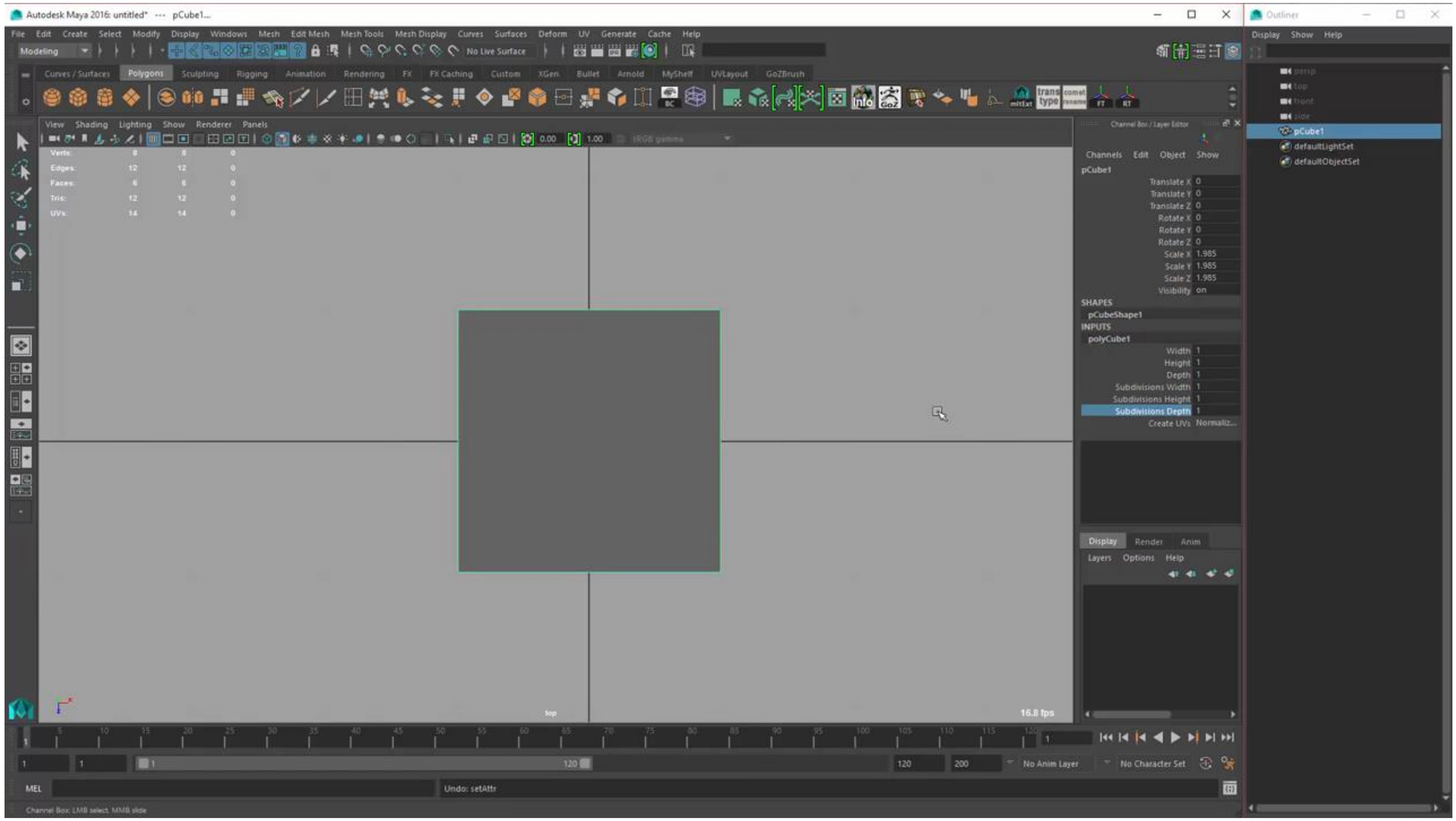
Sources of geometry

- **Digital 3D modeling**



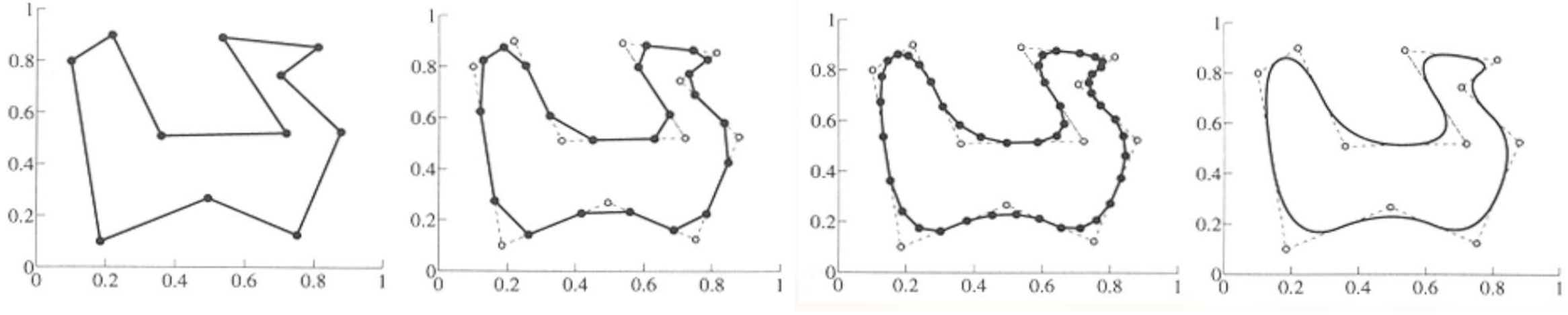
Sources of geometry

- **Digital 3D modeling**



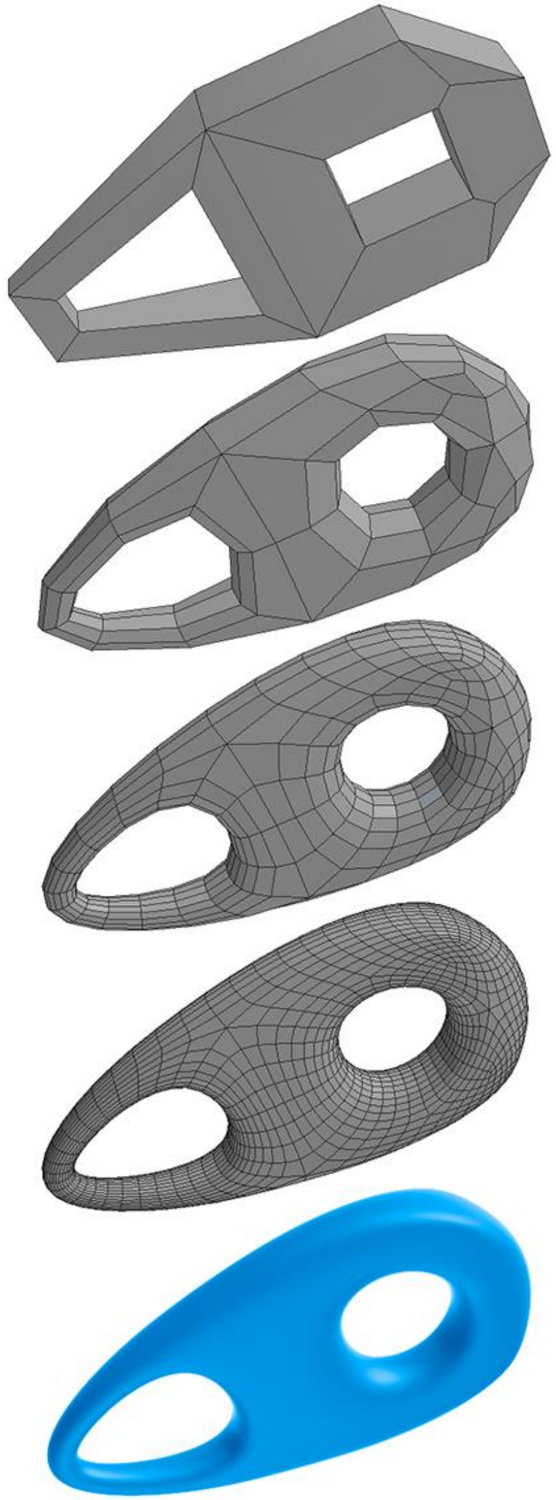
Subdivision Surfaces (Explicit)

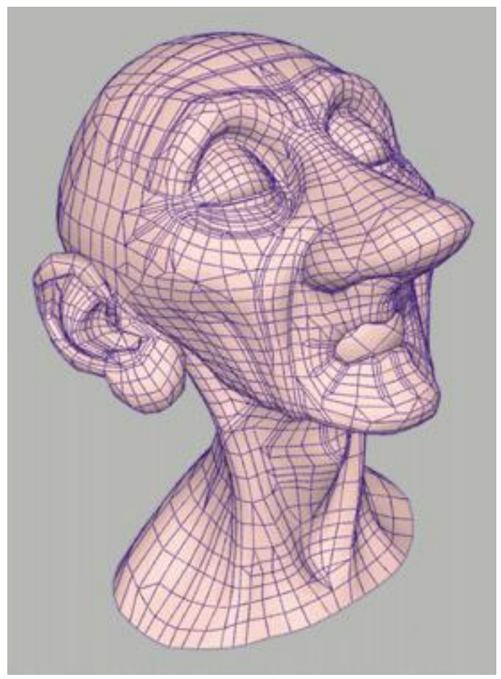
- Smooth out a control curve
  - Insert new vertex at each edge midpoint
  - Update vertex positions according to fixed rule
  - For careful choice of averaging rule, yields smooth curve
    - E.g. average with “next” neighbor (Chaikin)



Subdivision Surfaces (Explicit)

- Start with coarse polygon mesh (“control cage”)
- Subdivide each element
- Update vertices via local averaging
- Many possible rule:
  - Catmull-Clark (quads)
  - Loop (triangles)
  - …
- Common issues:
  - interpolating or approximating?
  - continuity at vertices?





Subdivision in Action (Pixar's "Geri's Game")