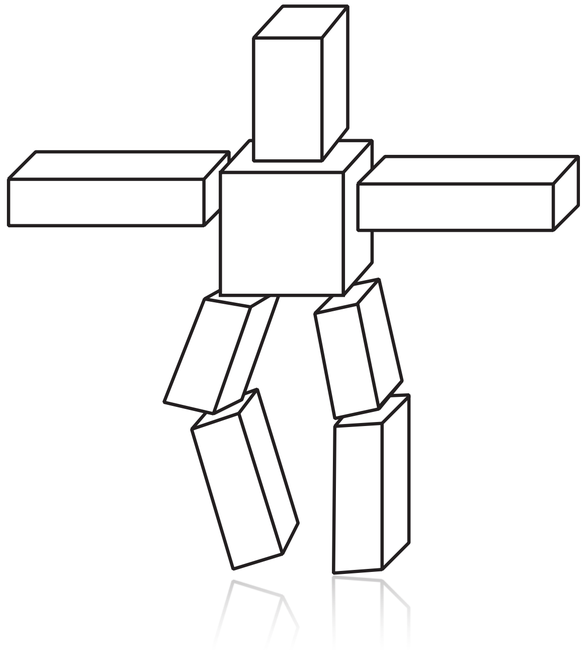
Lecture 4:

Representations of Geometry

Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2026

Increasing the complexity of our model of the world

Transformations (last time)



Geometry representations (today)

Materials, lighting, … (in the future)

Examples of geometry

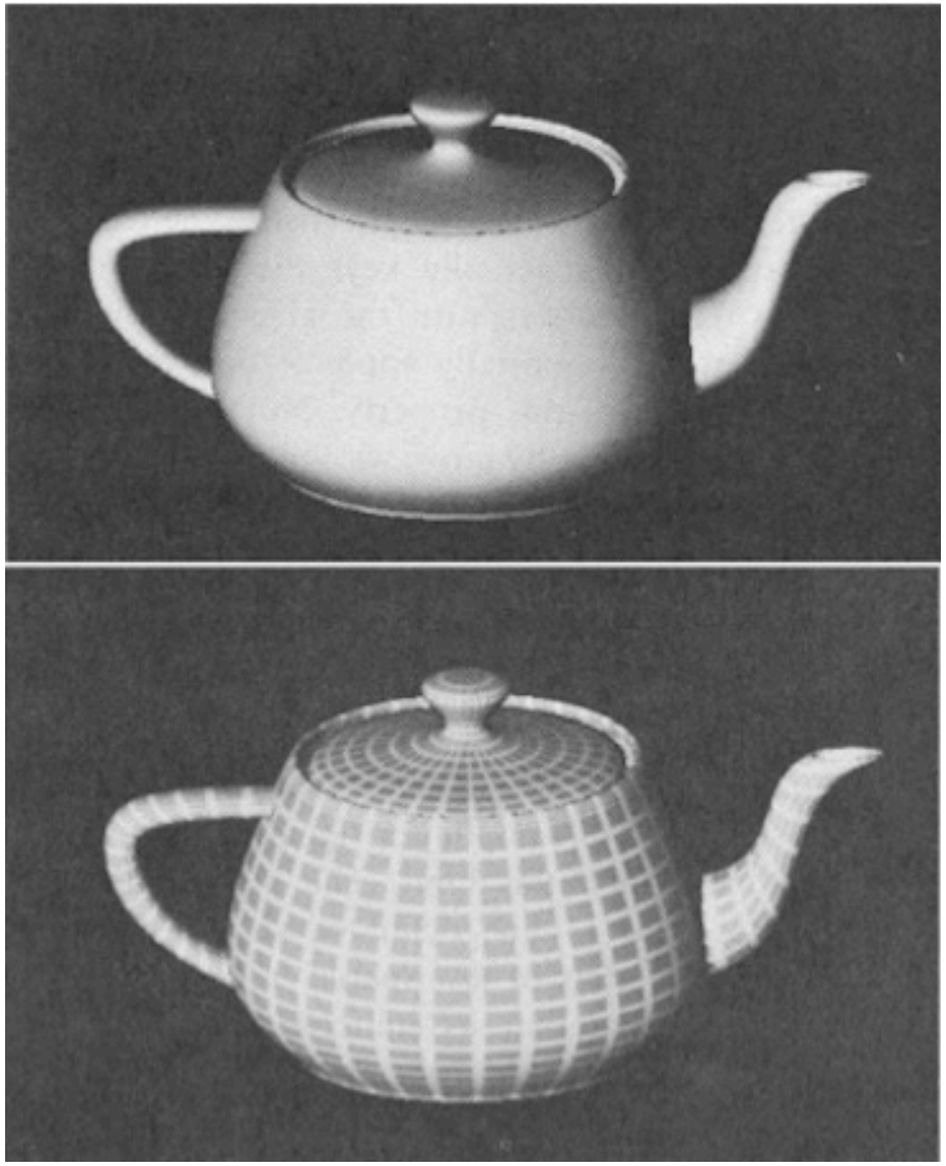


Photo of original Utah teapot
(now sitting in Computer History Museum in Mountain View)

Martin Newell's early teapot renderings
(Martin created teapot model in 1975 using Bezier curves)

Examples of geometry

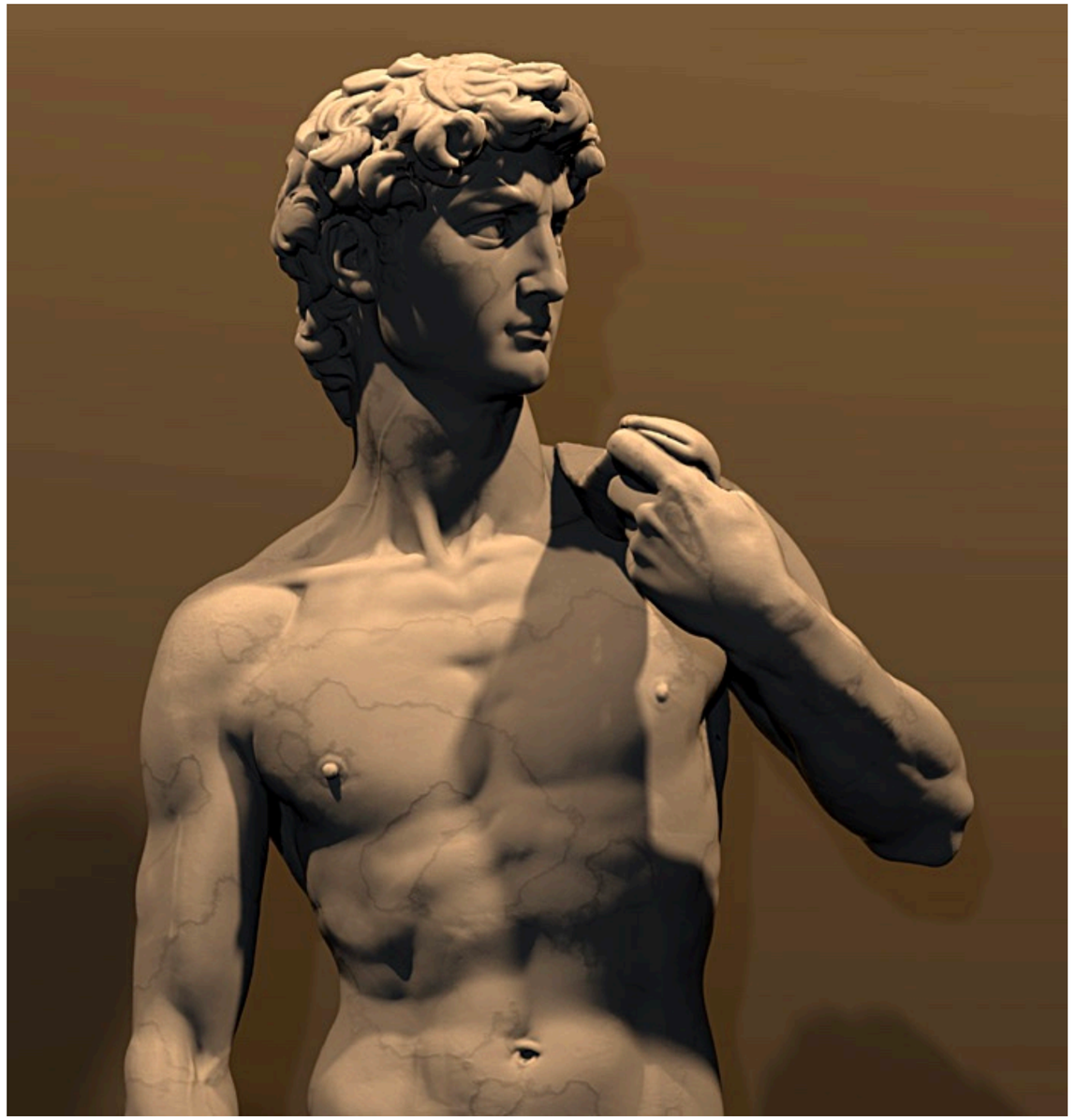
Cornell Box: Originally created in 1984
(This image was rendered in 1985 by Cohen and Greenberg)

Examples of geometry

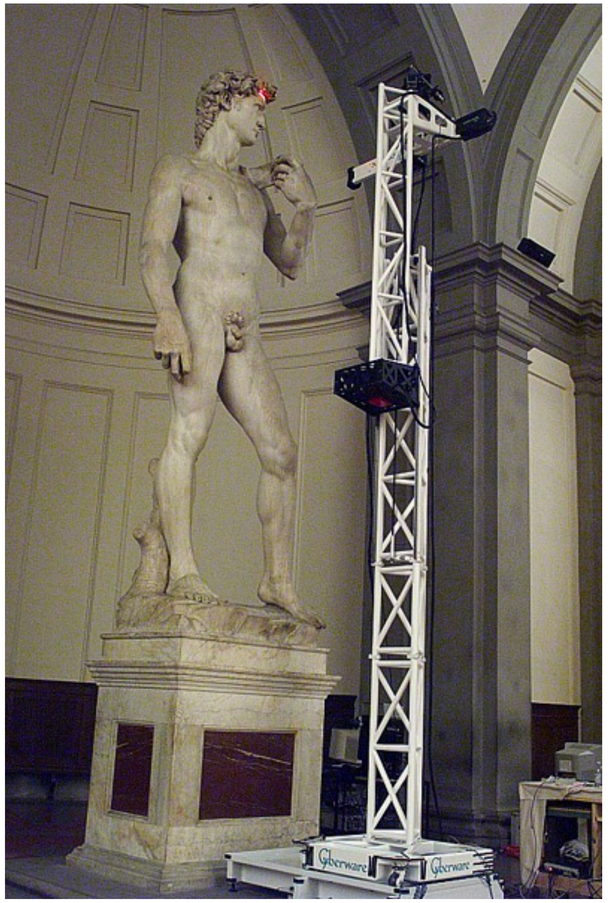
The Stanford Bunny
(Mesh created by reconstruction from laser scans)

Photograph of scanned statue
(Statue purchased by Greg Turk at
a store on University Ave in 1994)

Examples of geometry



Laser scan of Michelangelo's David
(Stanford's Digital Michelangelo project, 1999)



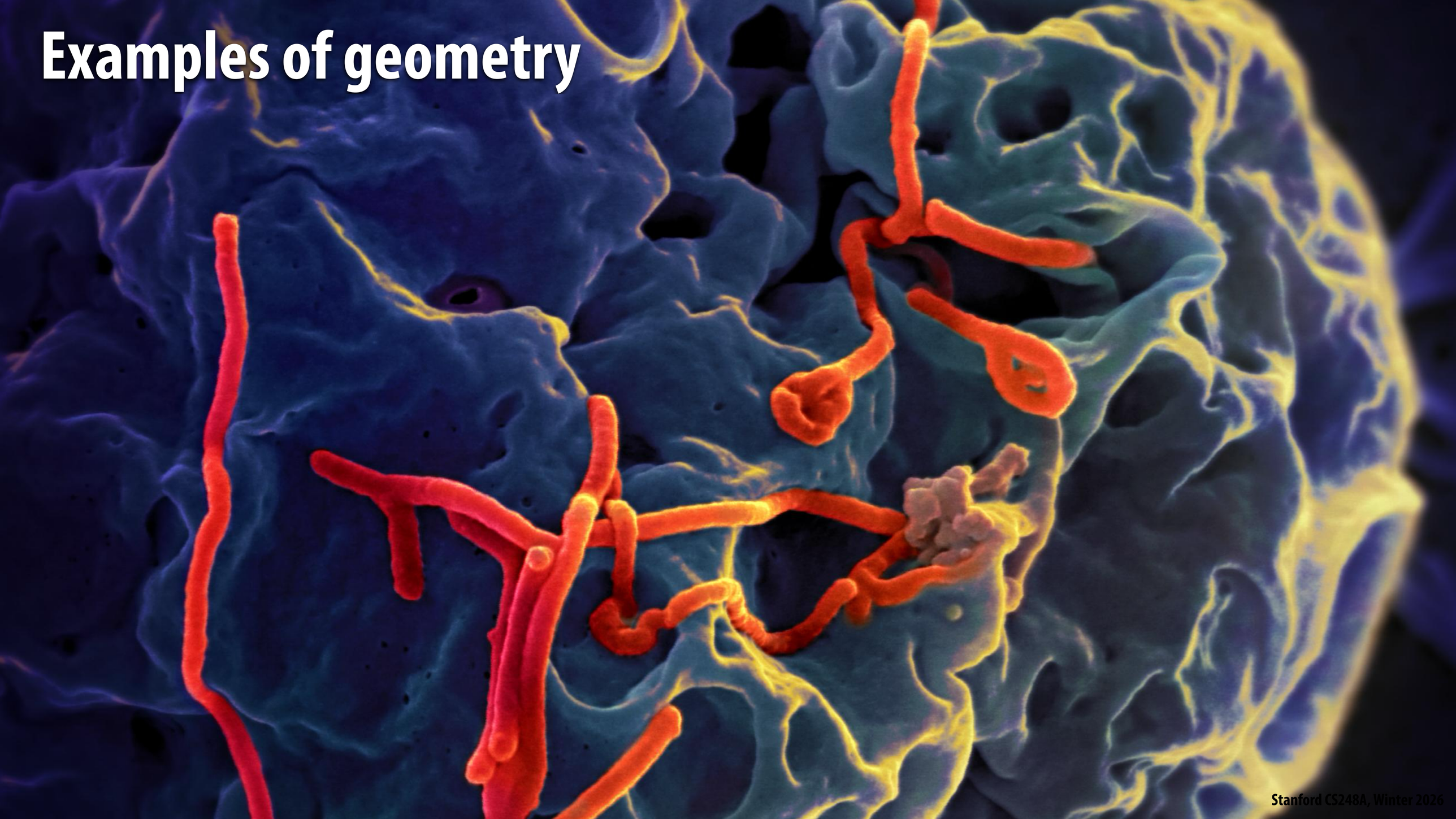
Examples of geometry

Photo credit:Natasha Moustache/TAS23

Examples of geometry

Examples of geometry

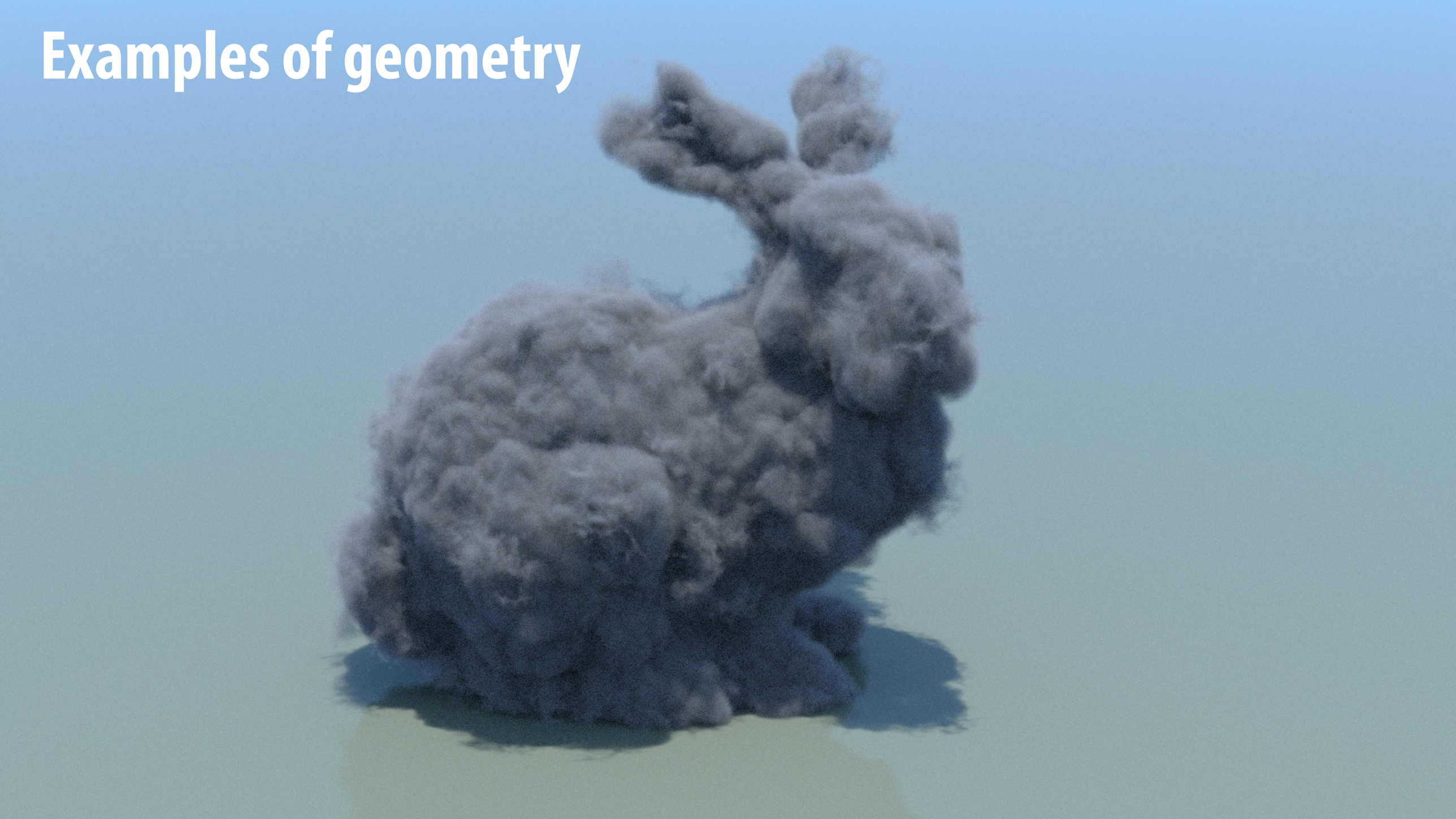
Examples of geometry



Examples of geometry

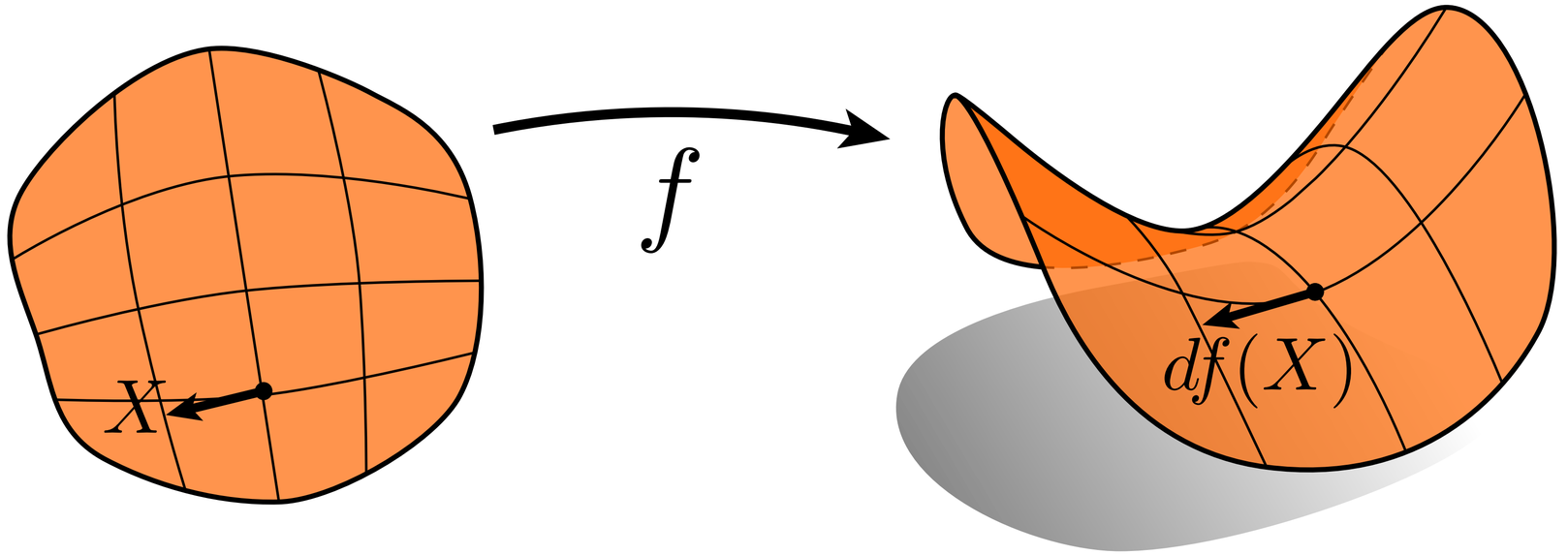
Curly hair in "Brave" (2012)

Examples of geometry



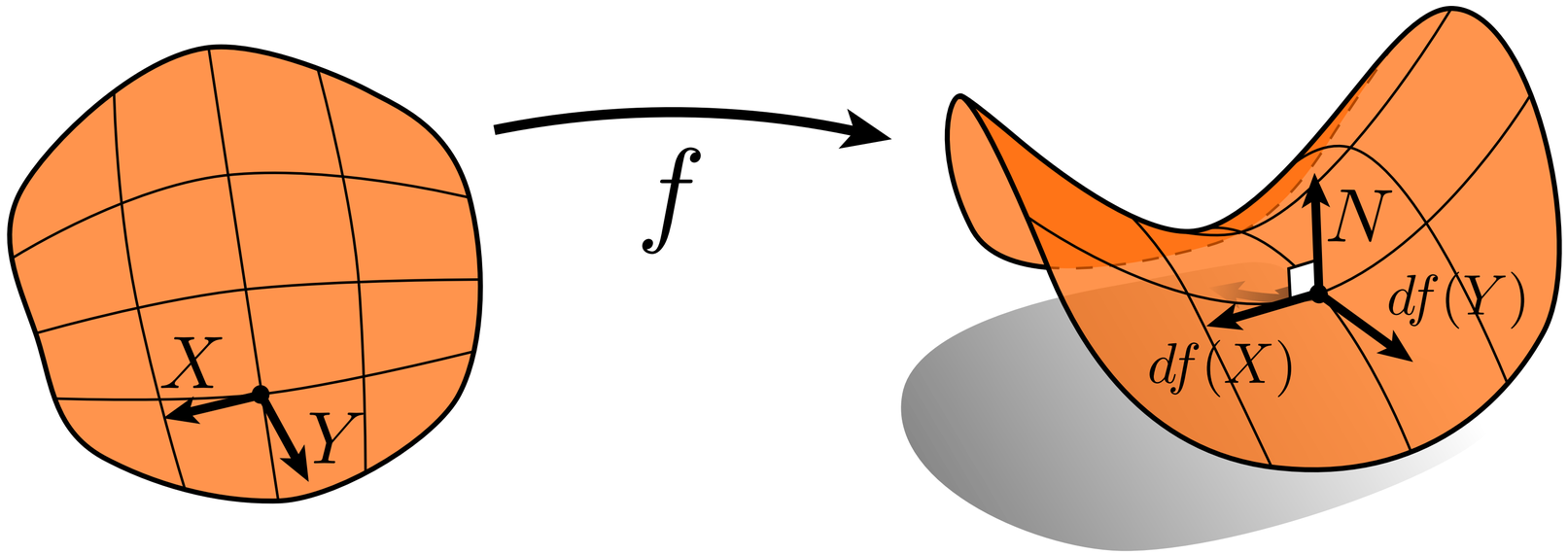
Measurements of surfaces

Surface tangent



Surface normal (N) is orthogonal to all tangents

$$N \cdot df(X) = 0 \quad \forall X$$



A common visualization of normals

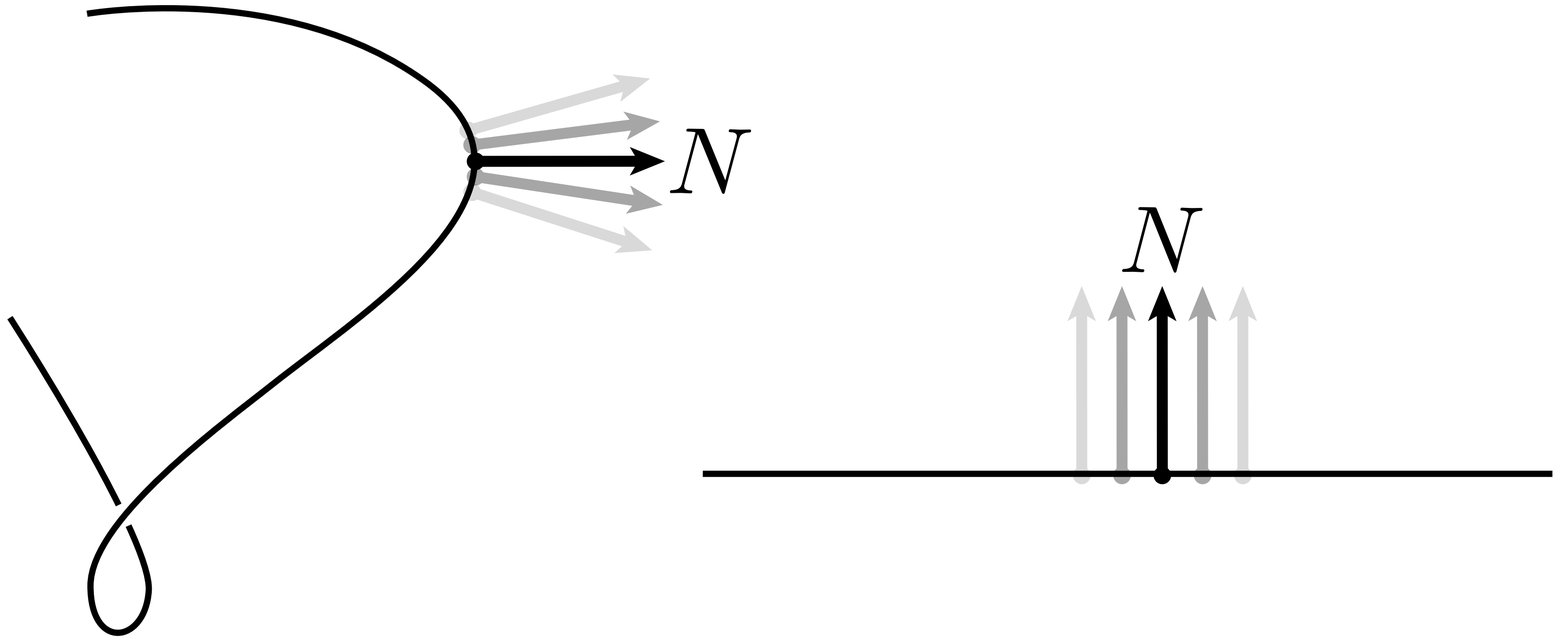
Encode normal direction as RGB color as difference from gray

$R = 0.5 + 0.5\ N.x$
$G = 0.5 + 0.5\ N.y$
$B = 0.5 + 0.5\ N.z$

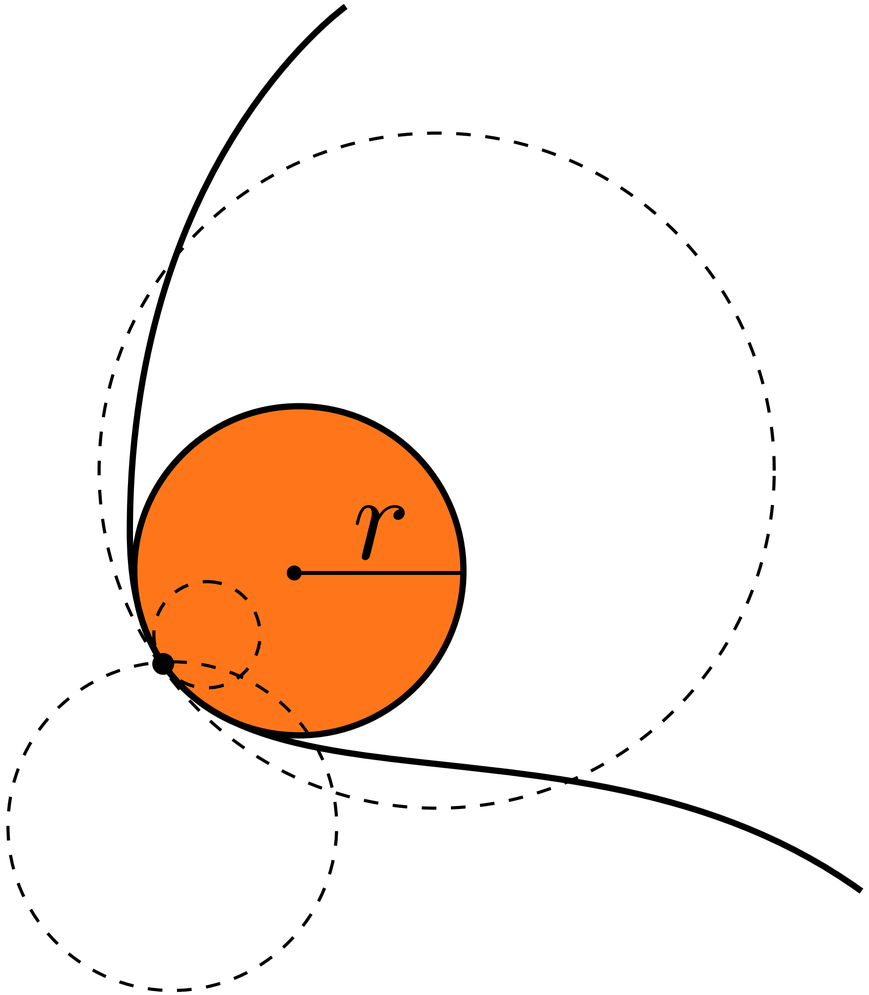
Notice: scale and bias normal values so we can represent negative components of normal as valid colors

Image credit: https://www.3dgep.com/forward-plus/

Curvature is *change* in normal



Radius of curvature



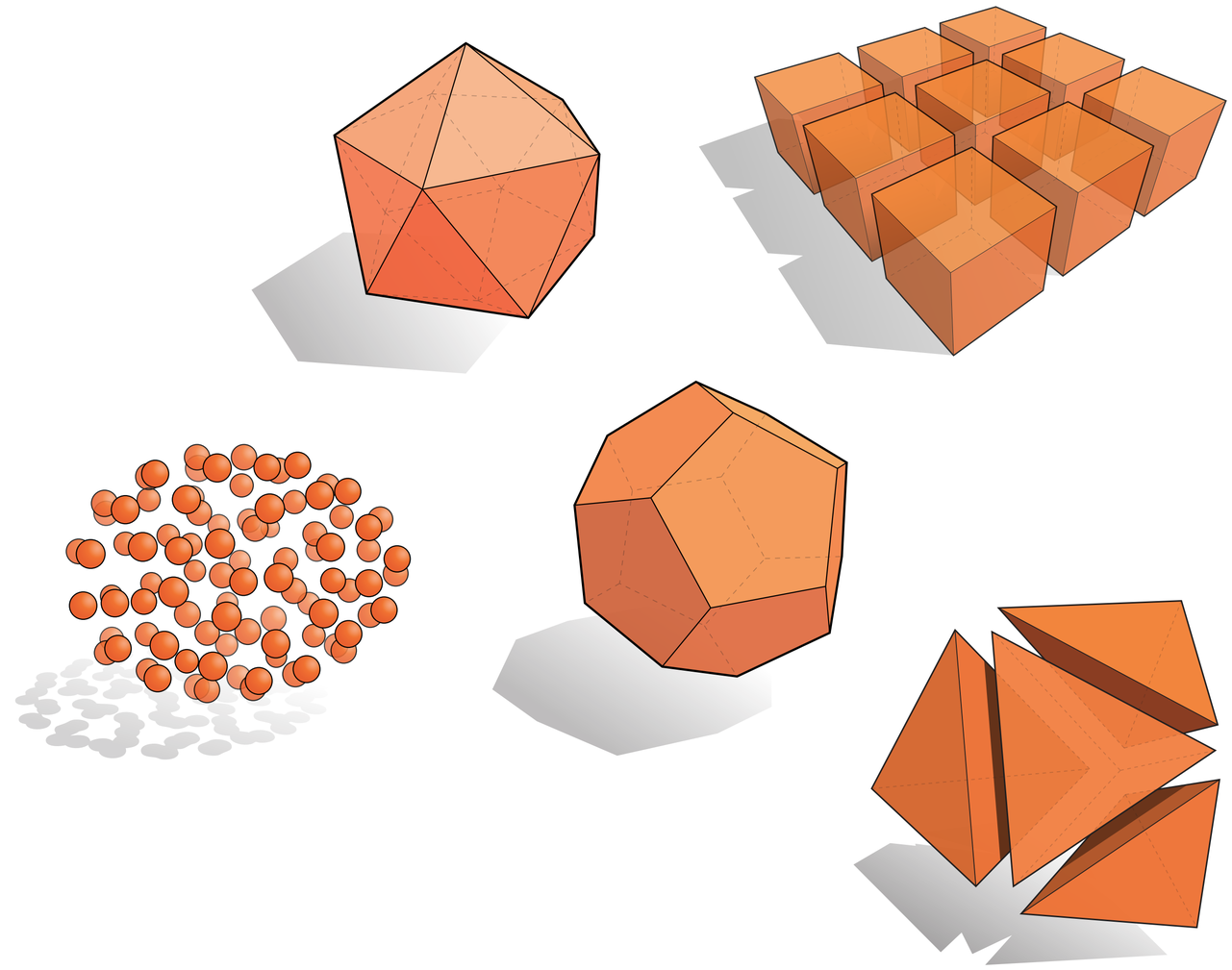
$$\kappa = \frac{1}{r}$$

curvature

What are ways to encode geometry on a computer?

Many ways to digitally encode geometry

- **EXPLICIT**
  - point cloud
  - volume
  - polygon mesh
  - subdivision surface…
  - …
- **IMPLICIT**
  - algebraic surface
  - distance field
  - occupancy field
  - L-systems
- **Each choice best suited to a different tasks or types of geometry**



“Implicit” representations of an object

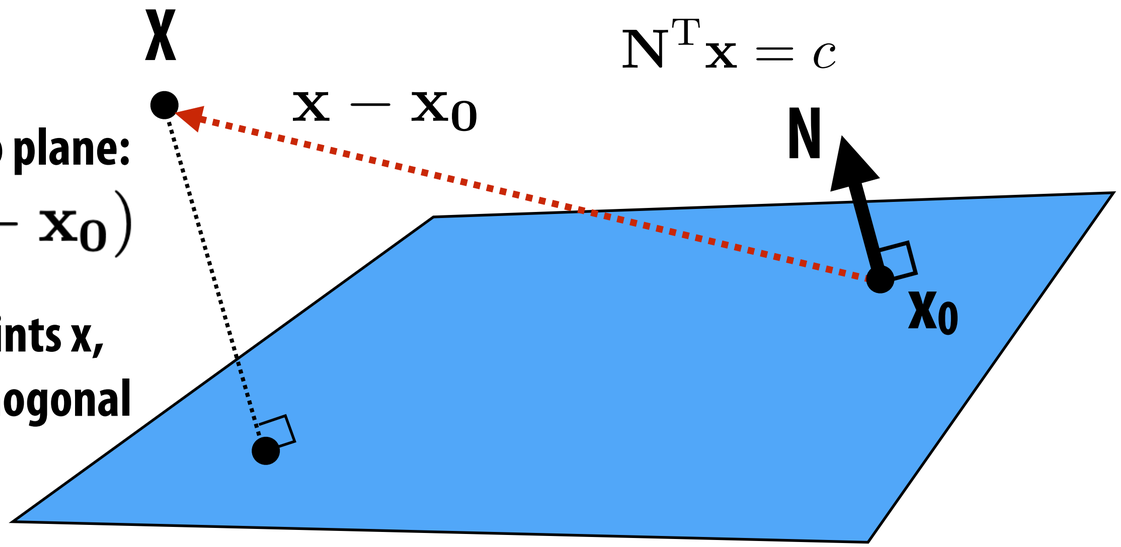
- Points on surface aren’t encoded directly, but satisfy a given relationship
- A plane is the set of points that satisfy $N^T x = c$
- Unit sphere centered at origin is set of point that satisfy $x^2+y^2+z^2=1$
- More generally, $f(x,y,z) = 0$

$$\mathbf{N} \cdot (\mathbf{x} - \mathbf{x_0}) = 0$$

$$\mathbf{N}^\mathrm{T}(\mathbf{x} - \mathbf{x_0}) = 0$$

$$\mathbf{N}^\mathrm{T}\mathbf{x} = \mathbf{N}^\mathbf{T}\mathbf{x_0}$$

$$\mathbf{N}^\mathrm{T}\mathbf{x} = c$$



Signed distance to plane:

$$\mathbf{N} \cdot (\mathbf{x} - \mathbf{x_0})$$

The plane is all points x, where x - x_0 is orthogonal to N.

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

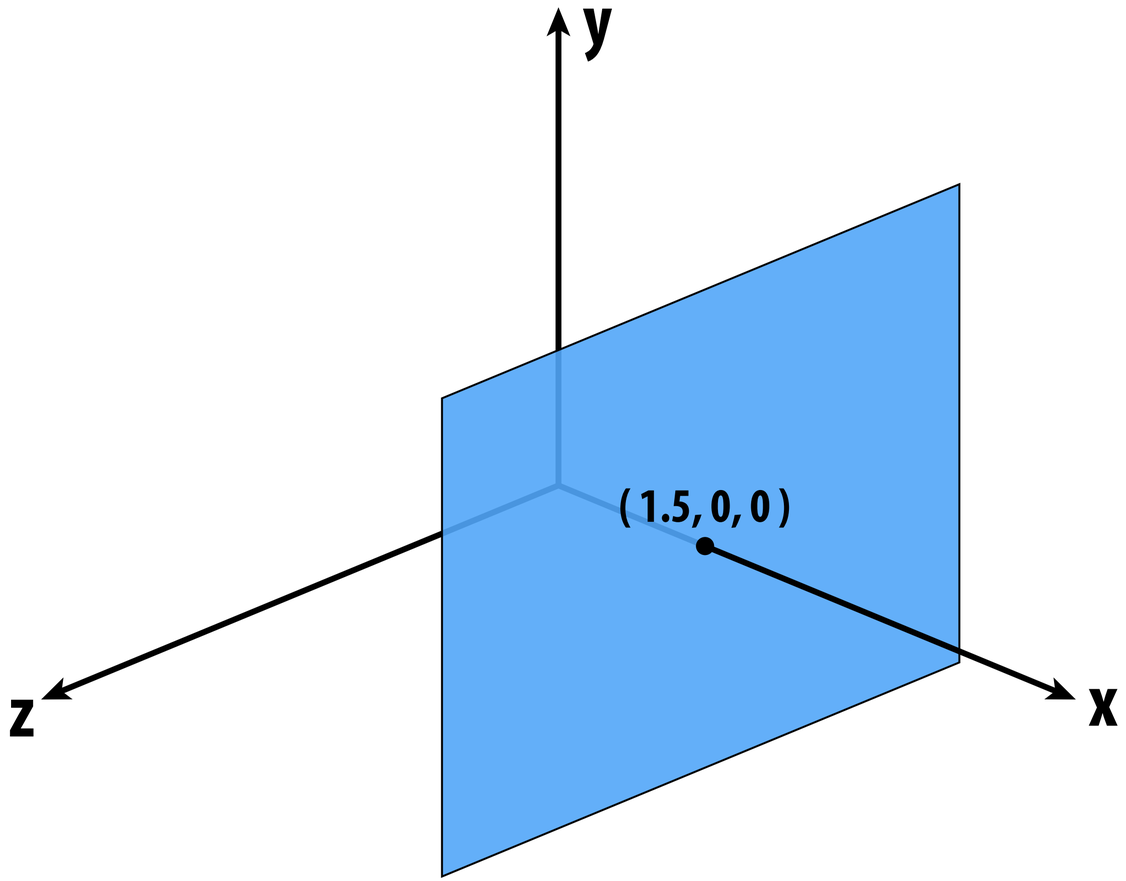
But first, let's play a game:

I'm thinking of an implicit surface f(x,y,z)=0

Find *any* point on it.

Give up?

My function was f(x,y,z) = x - 1.5 (a plane):



Implicit surfaces make some tasks hard (like sampling).

Let's play another game.

I have a new surface $f(x,y,z) = x^2 + y^2 + z^2 - 1$

I want to see if a point is *inside* it.

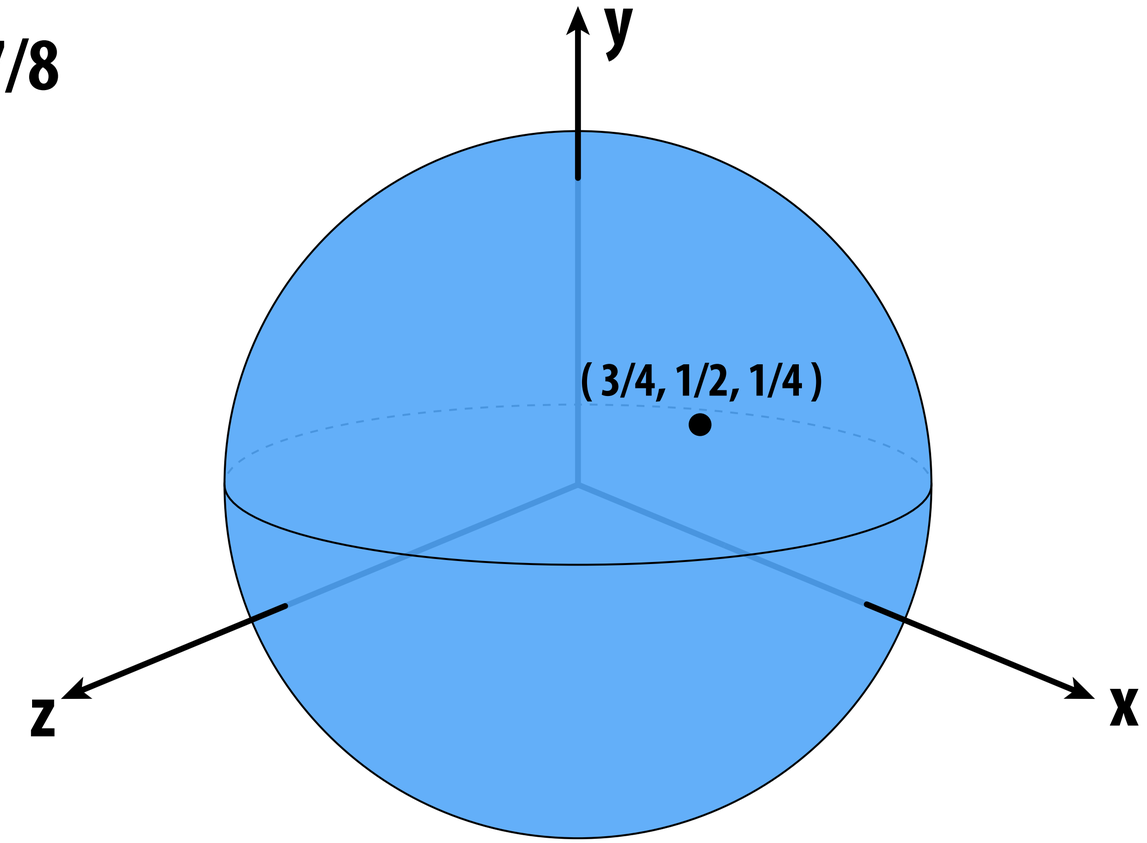
Check if this point is inside the unit sphere

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

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

$7/8 < 1$

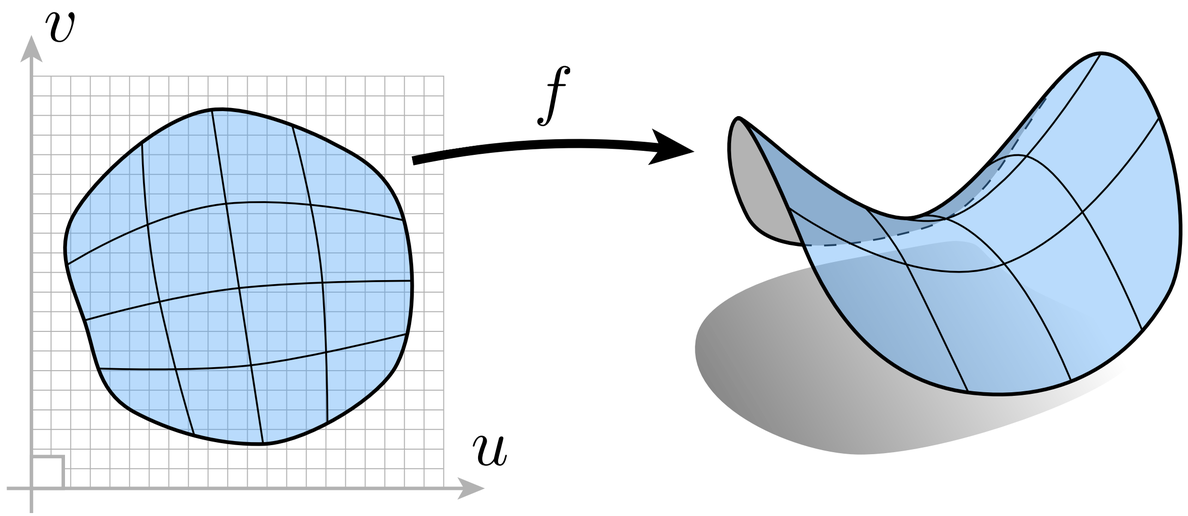
YES.



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

“Explicit” representations of geometry

- **All points are given directly (given a parameter representing a point on a surface… function provides a 3D point on the surface)**
- **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:** $f : \mathbb{R}^2 \to \mathbb{R}^3; (u, v) \mapsto (x, y, z)$



(Might have a bunch of these maps, e.g., one per triangle!)

Explicit representation of a ray

- Parameterized by distance from origin

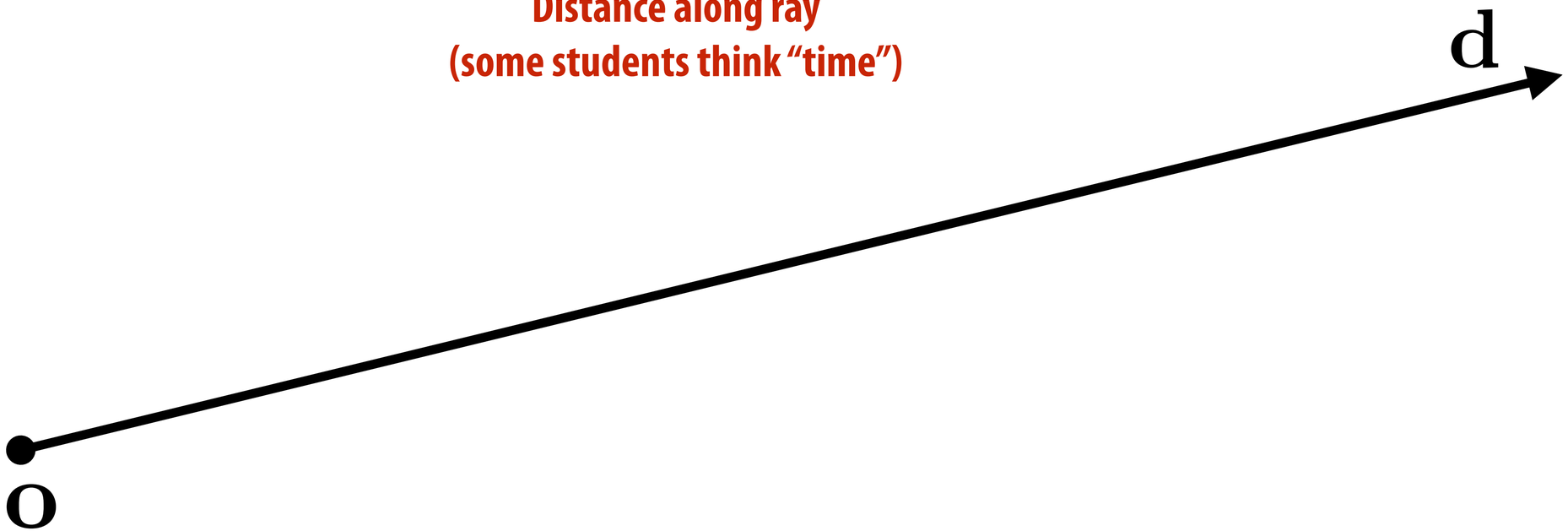
ray origin

unit direction

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

point along ray

Distance along ray
(some students think "time")



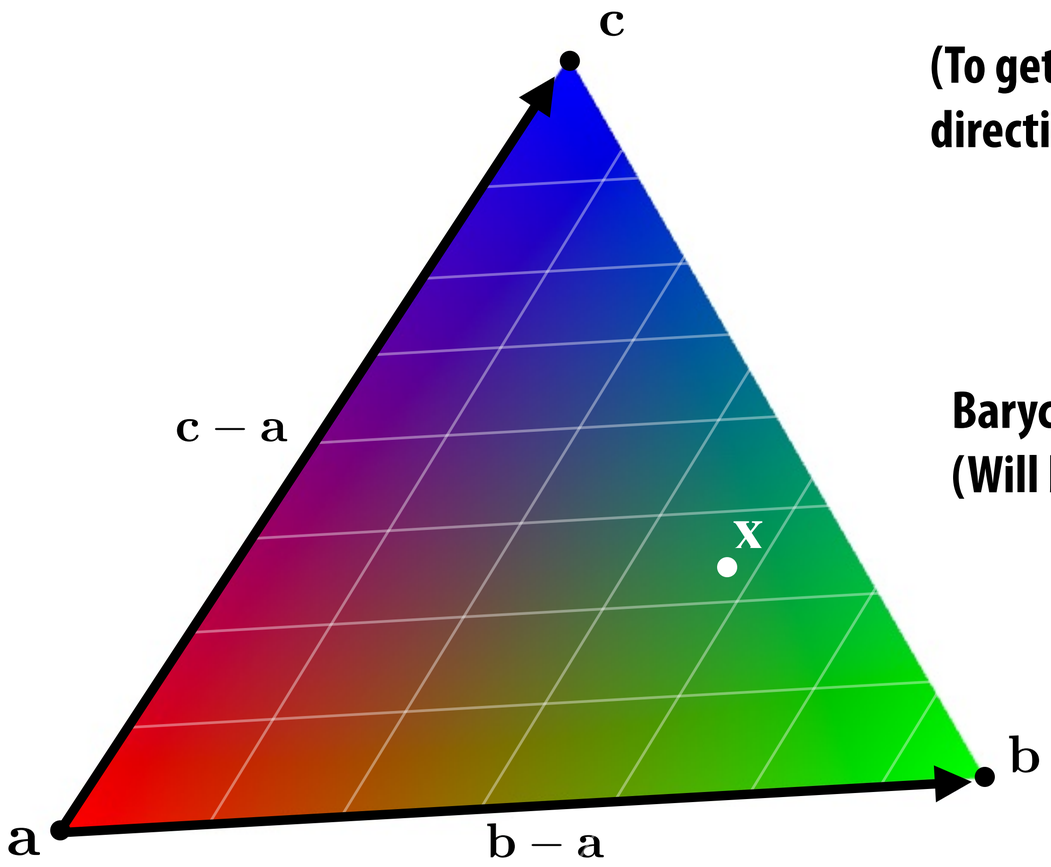
“Explicit” representation of a triangle

Given triangle vertices $\mathbf{a}, \mathbf{b}, \mathbf{c}$

$$\mathbf{x} = f(\beta, \gamma) = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

Triangle is parameterized by β and γ, where $\beta + \gamma < 1$

(To get point on triangle, move from vertex $\mathbf{a}$ β units in the $(\mathbf{b} - \mathbf{a})$ direction and γ gamma units in the $(\mathbf{c} - \mathbf{a})$ direction.)

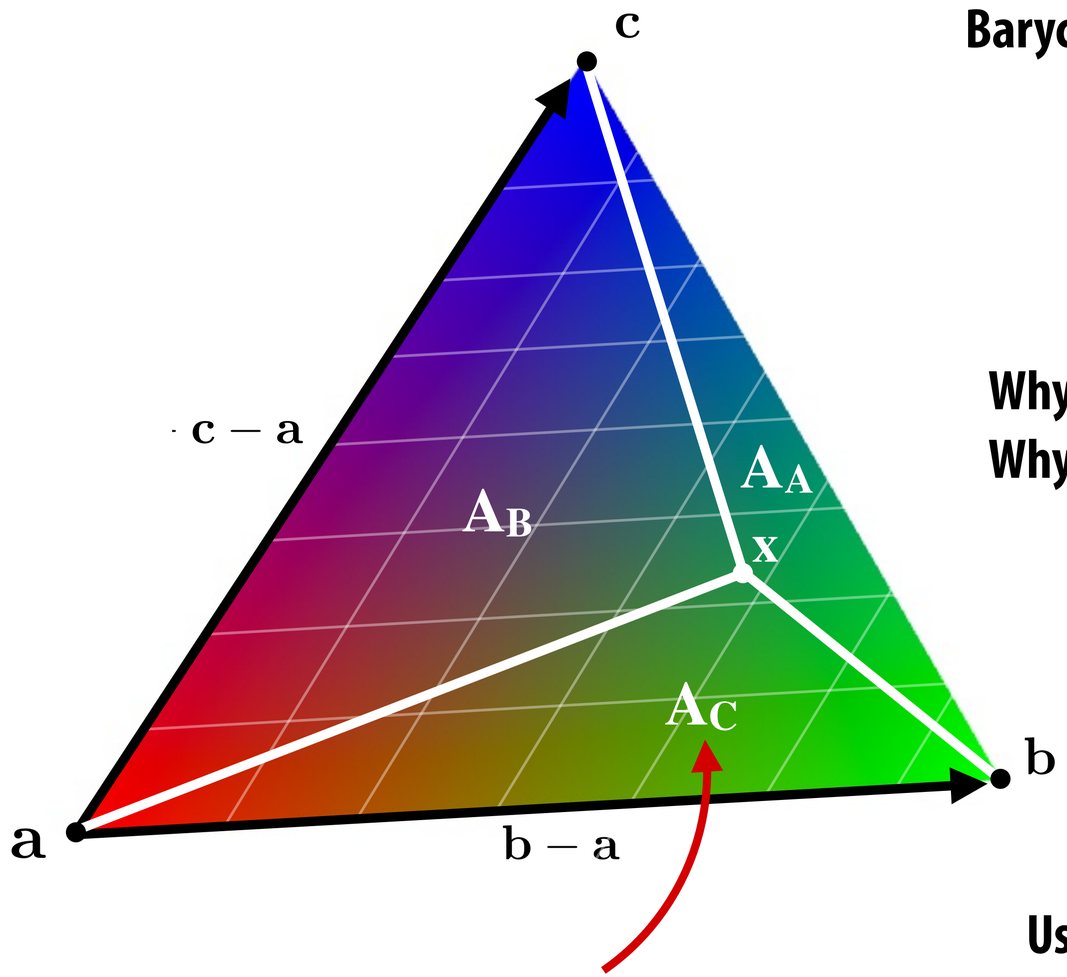


Barycentric parameterization of triangle in terms of: α, β, γ
(Will be useful parameterization in future texture mapping lecture.)

$$\begin{aligned}\mathbf{x} &= \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \\ &= (1 - \beta - \gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\ &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}\end{aligned}$$

$$\alpha + \beta + \gamma = 1$$

Barycentric coordinates (as a ratio of areas)



Area of triangle formed by points: a, b, x

Barycentric coords are *signed* areas:

$$\alpha = A_A/A$$

$$\beta = A_B/A$$

$$\gamma = A_C/A$$

Why must coordinates sum to one?

Why must coordinates be between 0 and 1?

Useful: Heron's formula:

$$A_C = \frac{1}{2}(\mathbf{b}-\mathbf{a}) \times (\mathbf{x}-\mathbf{a})$$

Another way to think about ray-triangle intersection

Plug parametric ray equation directly into equation for points on triangle:

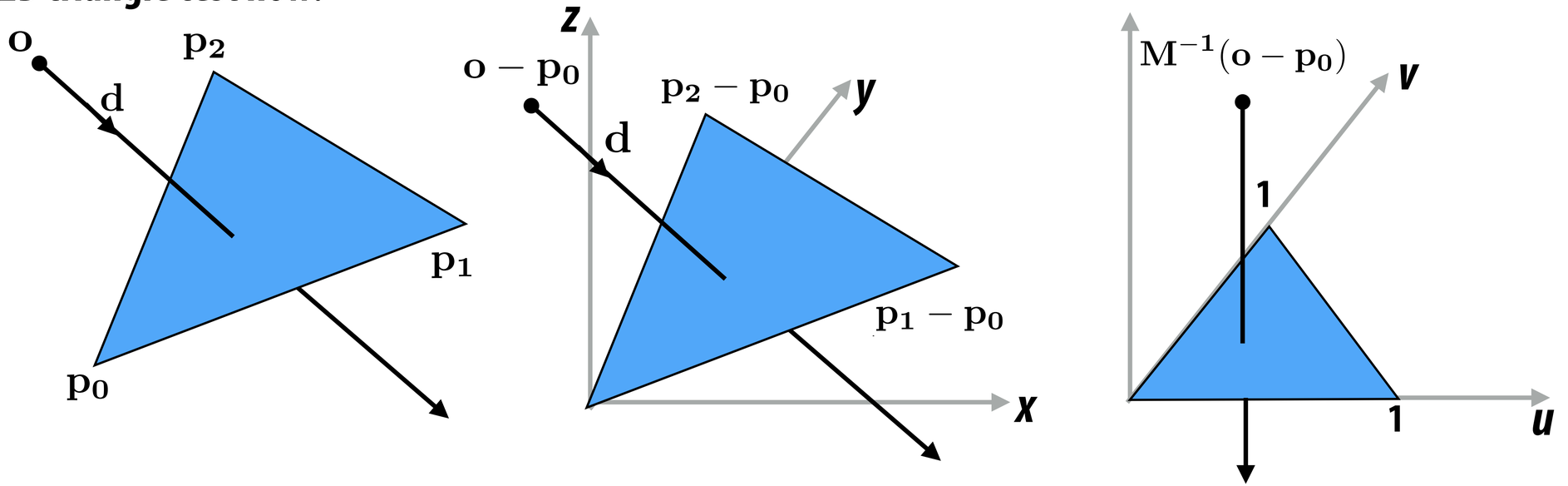
$$\mathbf{p_0} + u(\mathbf{p_1} - \mathbf{p_0}) + v(\mathbf{p_2} - \mathbf{p_0}) = \mathbf{o} + t\mathbf{d}$$

Solve for u, v, t:

$$\underbrace{\begin{bmatrix} \mathbf{p_1} - \mathbf{p_0} & \mathbf{p_2} - \mathbf{p_0} & -\mathbf{d} \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} u \\ v \\ t \end{bmatrix} = \mathbf{o} - \mathbf{p_0}$$

$\mathbf{M}^{-1}$ **transforms triangle back to unit triangle in u,v plane, and transforms ray's direction to be orthogonal to plane.**

It's a point in 2D triangle test now!



But first, let's play another game:

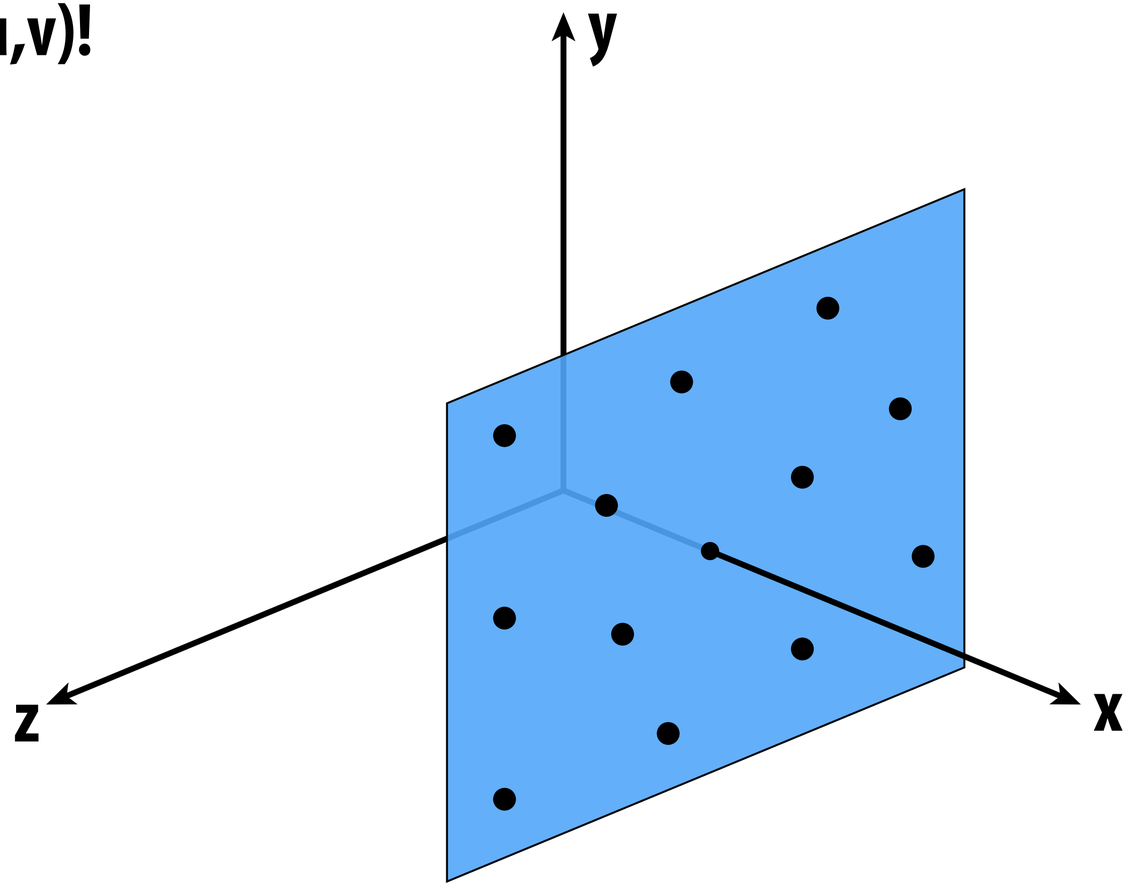
I'll give you an explicit surface.

You give me some points on it.

Sampling an explicit surface

My surface is f(u, v) = (1.5, u, v).

Just plug in any values (u,v)!



Explicit surfaces make some tasks easy (like sampling).

Let's play another game.

I have a new surface f(u,v).

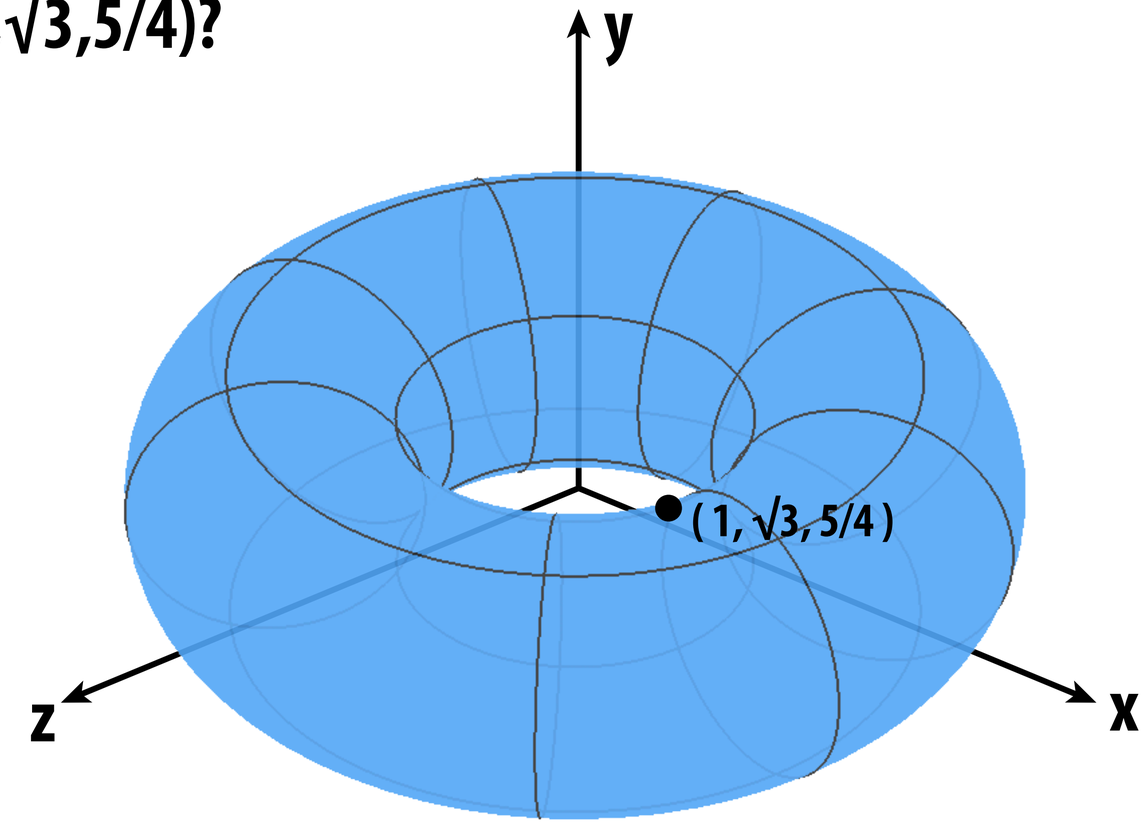
I want to see if a point is *inside* it.

Check if this point is inside the torus

My surface is f(u,v) = (2+cos(u))cos(v), 2+cos(u))sin(v), sin(u))

How about the point (1,√3,5/4)?

...NO!



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

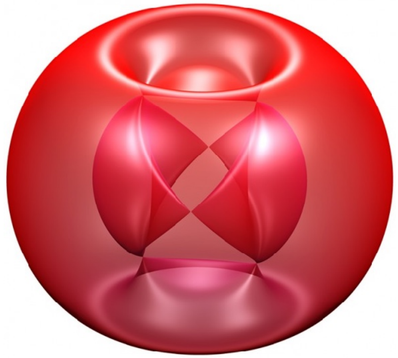
CONCLUSION:
Some representations work better than others—depending on the task!

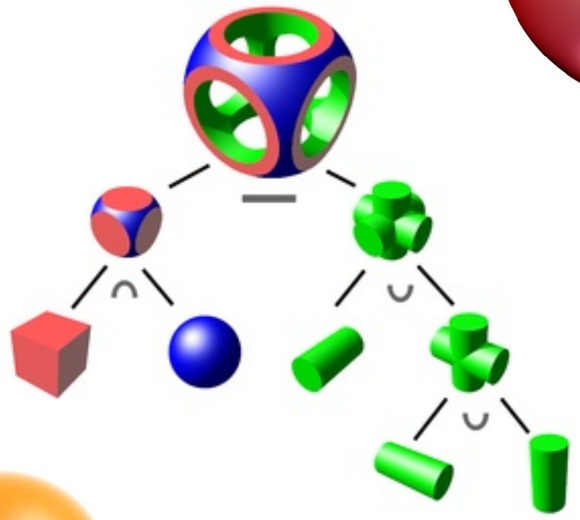
Different representations will be better suited to different types of geometry.

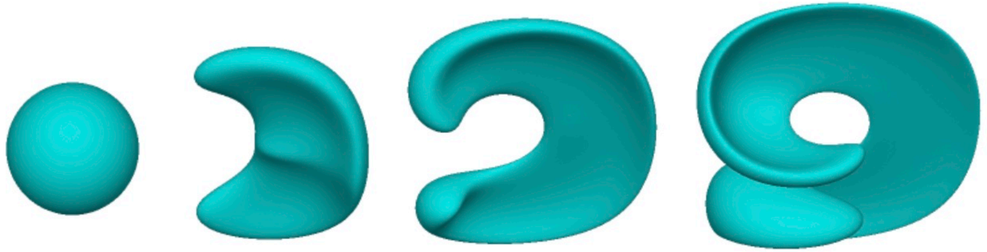
Let's take a look at some common representations used in computer graphics.

Many implicit representations in graphics

- algebraic surfaces
- constructive solid geometry
- level set methods
- blobby surfaces
- fractals
- ...

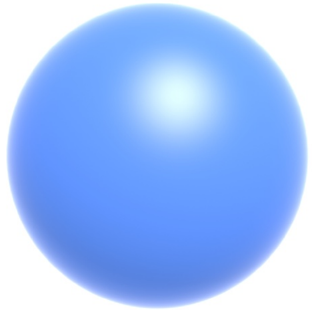




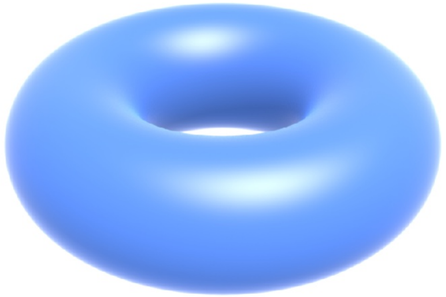


Algebraic surfaces (implicit)

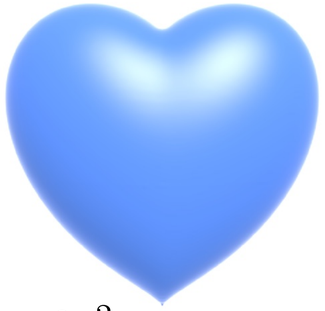
- Surface is zero set of a polynomial in x, y, z (“algebraic variety”)
- Examples:



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



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



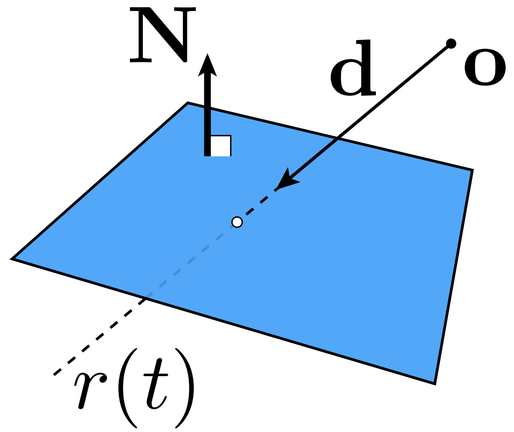
$$(x^2+\frac{9y^2}{4}+z^2-1)^3=x^2z^3+\frac{9y^2z^3}{80}$$

- What about more complicated shapes?

- Hard to come up with polynomials for complex shapes!

Example task: intersection of ray with implicit

- Key idea: find point on ray that satisfies properties of the implicit surface definition



- Example from lecture 1: ray-plane intersection
- Suppose we have a plane $\mathbf{N}^T\mathbf{x} = c$
- Replace the point x in the implicit equation with the ray equation parameterized by t:

$$\mathbf{N}^\mathsf{T}(\mathbf{o} + t\mathbf{d}) = c$$

- Now solve for t:

$$\Rightarrow t = \frac{c - \mathbf{N}^\mathsf{T}\mathbf{o}}{\mathbf{N}^\mathsf{T}\mathbf{d}}$$

- *t* is the distance to the plane from the ray origin!

And the "hit point" on the plane is:

$$r(t) = \mathbf{o} + \frac{c - \mathbf{N}^\mathsf{T}\mathbf{o}}{\mathbf{N}^\mathsf{T}\mathbf{d}}\mathbf{d}$$

Ray- (unit) sphere intersection

What points on the ray satisfy the implicit equation for the unit sphere?

$$f(\mathbf{x}) = |\mathbf{x}|^2 - 1$$

$$\Rightarrow f(\mathbf{r}(t)) = |\mathbf{o} + t\mathbf{d}|^2 - 1$$

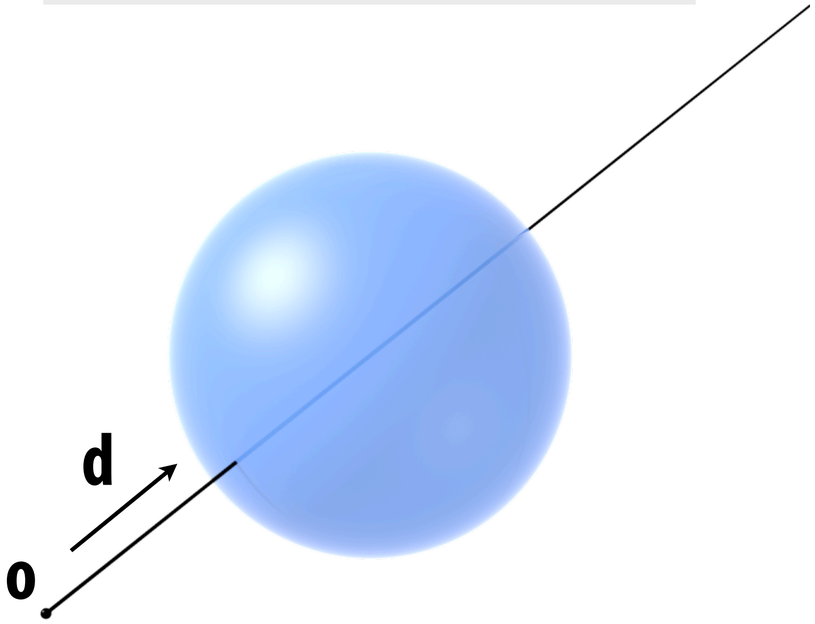
$$\underbrace{|\mathbf{d}|^2}_{a}\, t^2 + \underbrace{2(\mathbf{o}\cdot\mathbf{d})}_{b}\, t + \underbrace{|\mathbf{o}|^2 - 1}_{c} = 0$$

Note: $|\mathbf{d}|^2 = 1$ **since d is a unit vector**

$$t = \boxed{-\mathbf{o}\cdot\mathbf{d} \pm \sqrt{(\mathbf{o}\cdot\mathbf{d})^2 - |\mathbf{o}|^2 + 1}}$$

Recall the quadratic formula:

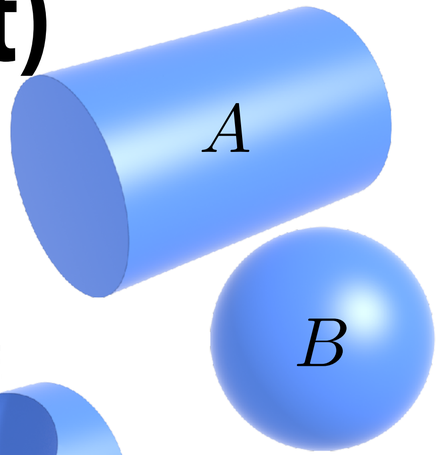
$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



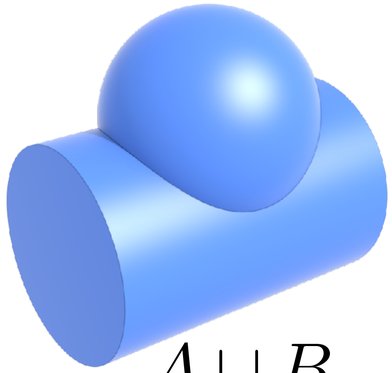
Why two solutions?

Constructive solid geometry (CSG) (implicit)

- Build more complicated shapes using boolean operations
- Basic operations on volumes:



UNION

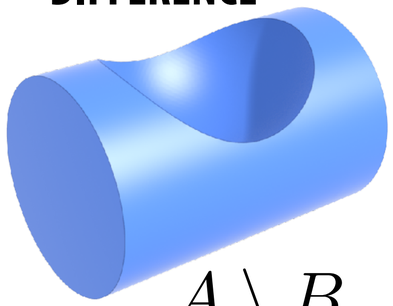


$A \cup B$

INTERSECTION

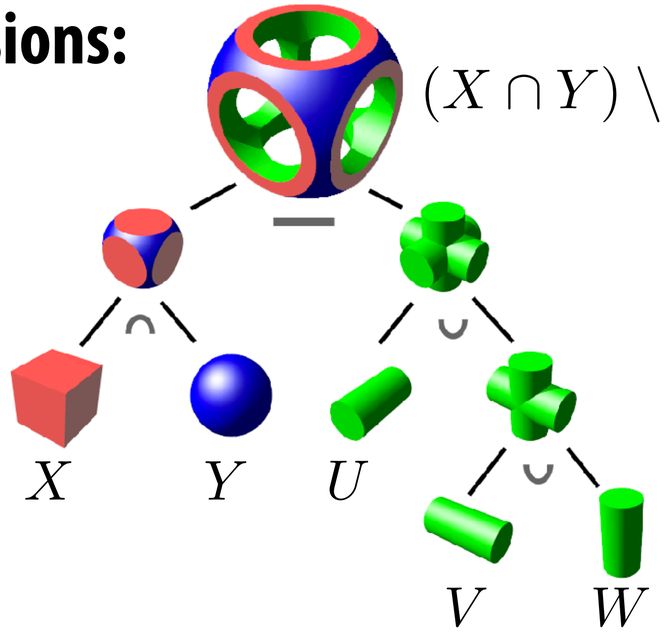
$A \cap B$

DIFFERENCE



$A \setminus B$

- Then build more complex expressions:

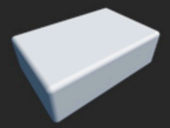


Signed distance functions (SDF)

Sphere - exact

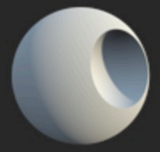
https://www.shadertoy.com/view/Xds3zN

```
float sdSphere( vec3 p, float s )
{
  return length(p)-s;
}
```



Round Box - exact

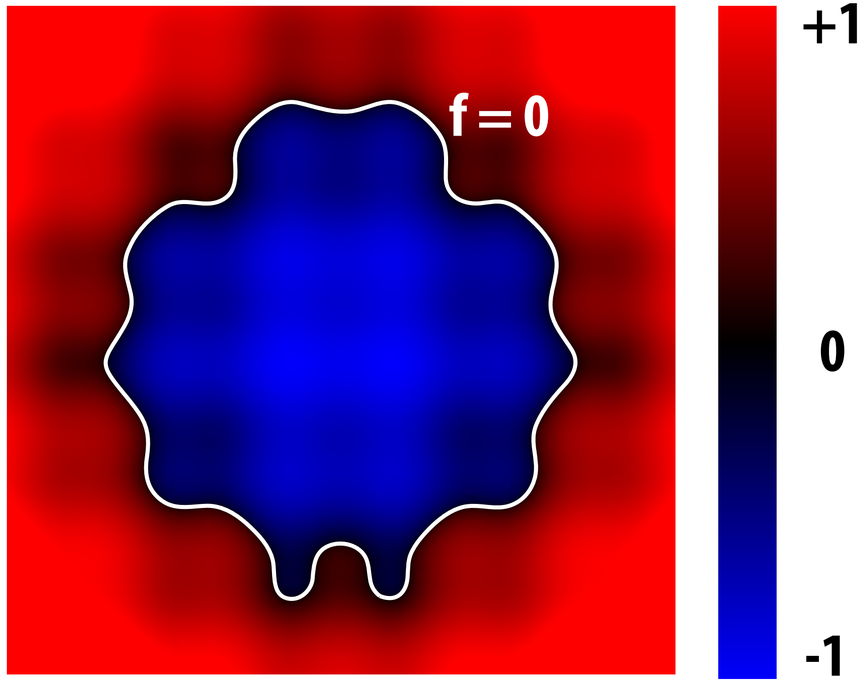
```
float sdRoundBox( vec3 p, vec3 b, float r )
{
  vec3 q = abs(p) - b + r;
  return length(max(q,0.0)) + min(max(q.x,max(q.y,q.z)),0.0) - r;
}
```



Death Star - exact

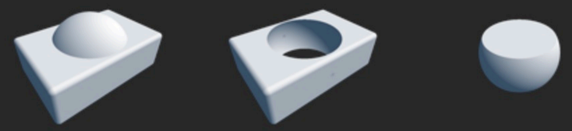
https://www.shadertoy.com/view/7lVXRt

```
float sdDeathStar( vec3 p2, float ra, float rb, float d )
{
  float a = (ra*ra - rb*rb + d*d)/(2.0*d);
  float b = sqrt(max(ra*ra-a*a,0.0));

  vec2 p = vec2( p2.x, length(p2.yz) );
  if( p.x*b-p.y*a > d*max(b-p.y,0.0) )
    return length(p-vec2(a,b));
  else
    return max( (length(p            )-ra),
               -(length(p-vec2(d,0.0))-rb));
}
```



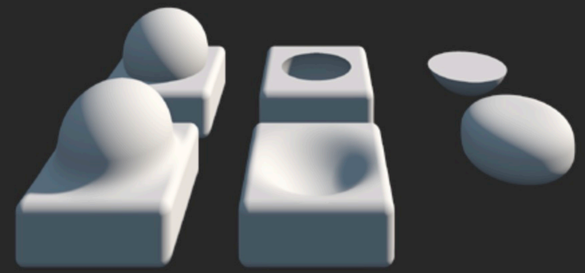
A great reference:
https://iquilezles.org/articles/distfunctions

SFD compositions



```
float opUnion( float d1, float d2 )
{
    return min(d1,d2);
}
float opSubtraction( float d1, float d2 )
{
    return max(-d1,d2);
}
float opIntersection( float d1, float d2 )
{
    return max(d1,d2);
}
float opXor( float d1, float d2 )
{
    return max(min(d1,d2),-max(d1,d2));
}
```

```
float opRepetition( in vec3 p, in vec3 s, in sdf3d primitive )
{
    vec3 q = p - s*round(p/s);
    return primitive( q );
}
```



```
float opSmoothUnion( float d1, float d2, float k )
{
    k *= 4.0;
    float h = max(k-abs(d1-d2),0.0);
    return min(d1, d2) - h*h*0.25/k;
}

float opSmoothSubtraction( float d1, float d2, float k )
{
    return -opSmoothUnion(d1,-d2,k);

    // k *= 4.0;
    // float h = max(k-abs(-d1-d2),0.0);
    // return max(-d1, d2) + h*h*0.25/k;
}

float opSmoothIntersection( float d1, float d2, float k )
{
    return -opSmoothUnion(-d1,-d2,k);

    // k *= 4.0;
    // float h = max(k-abs(d1-d2),0.0);
    // return max(d1, d2) + h*h*0.25/k;
}
```

A great reference:
https://iquilezles.org/articles/distfunctions

Scene of pure distance functions (not easy!)

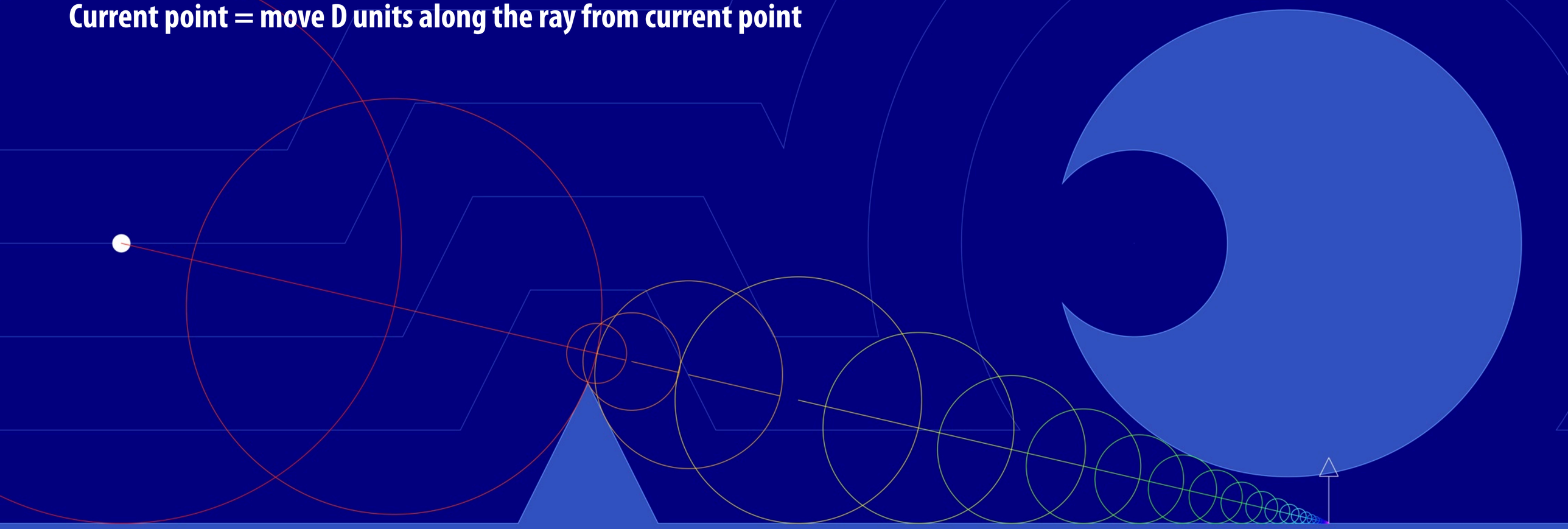
Image Credit: Inigo Quilez, https://www.shadertoy.com/view/ld3Gz2

Scene of pure distance functions (not easy!)

Image Credit: Inigo Quilez, https://www.shadertoy.com/view/WsSBzh

Example task: intersect ray with scene containing SDFs

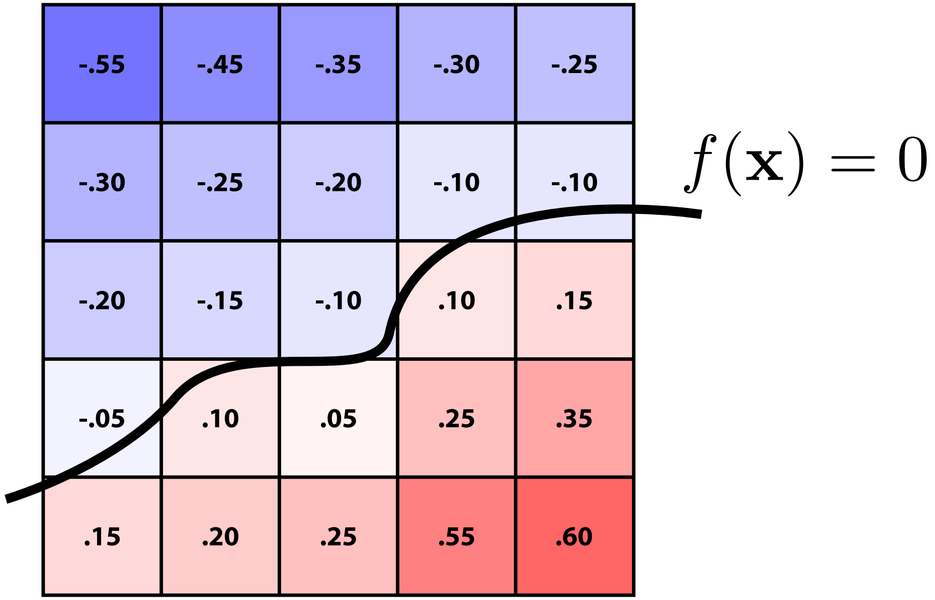
Current point = ray origin
While (distance D from current point to closest point on surface is not 0) // while not at surface
  Current point = move D units along the ray from current point



Credit: https://www.shadertoy.com/view/lslXD8

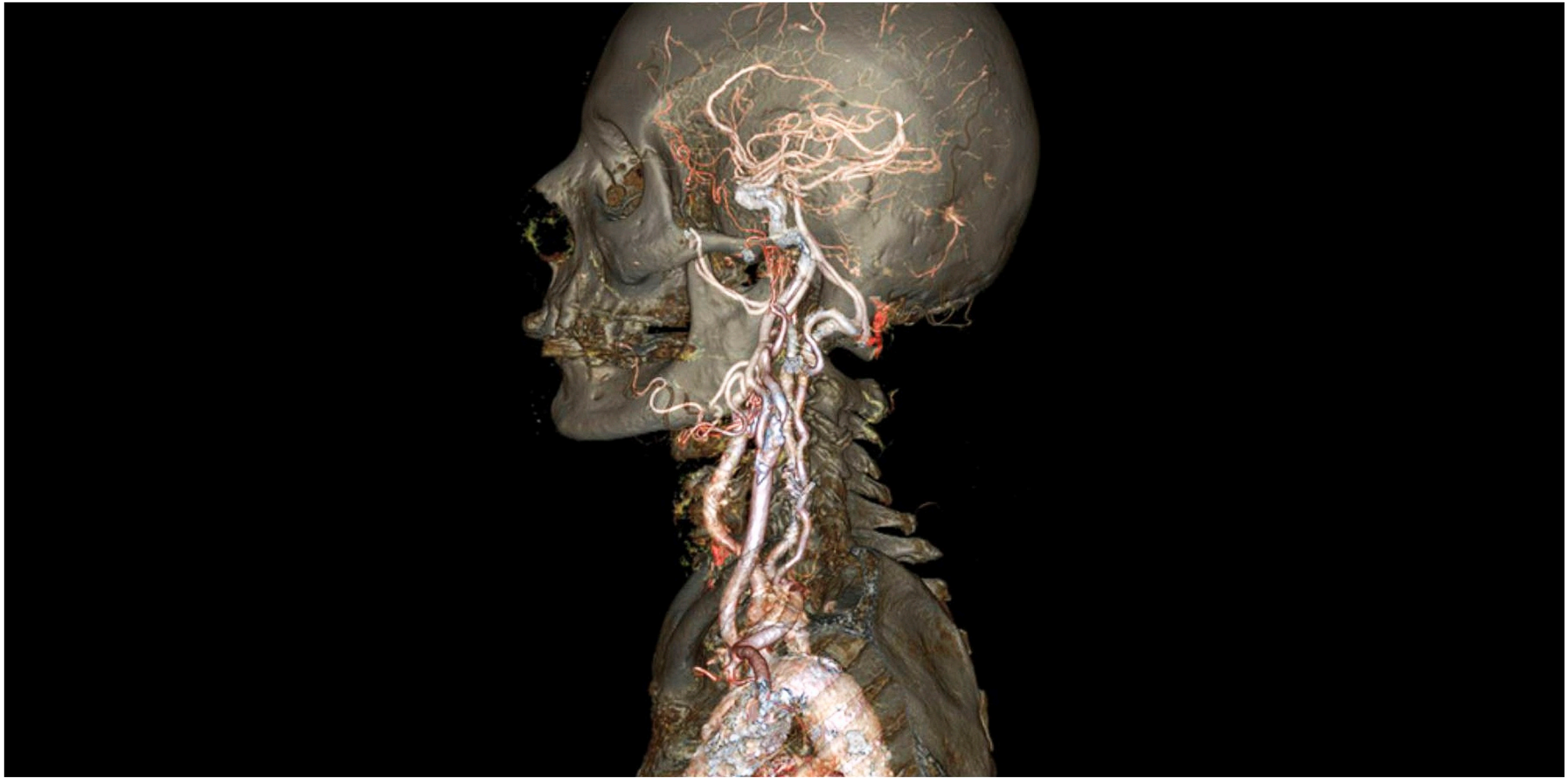
Distance function represented as dense samples

- 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 a continuous function (samples of the function)



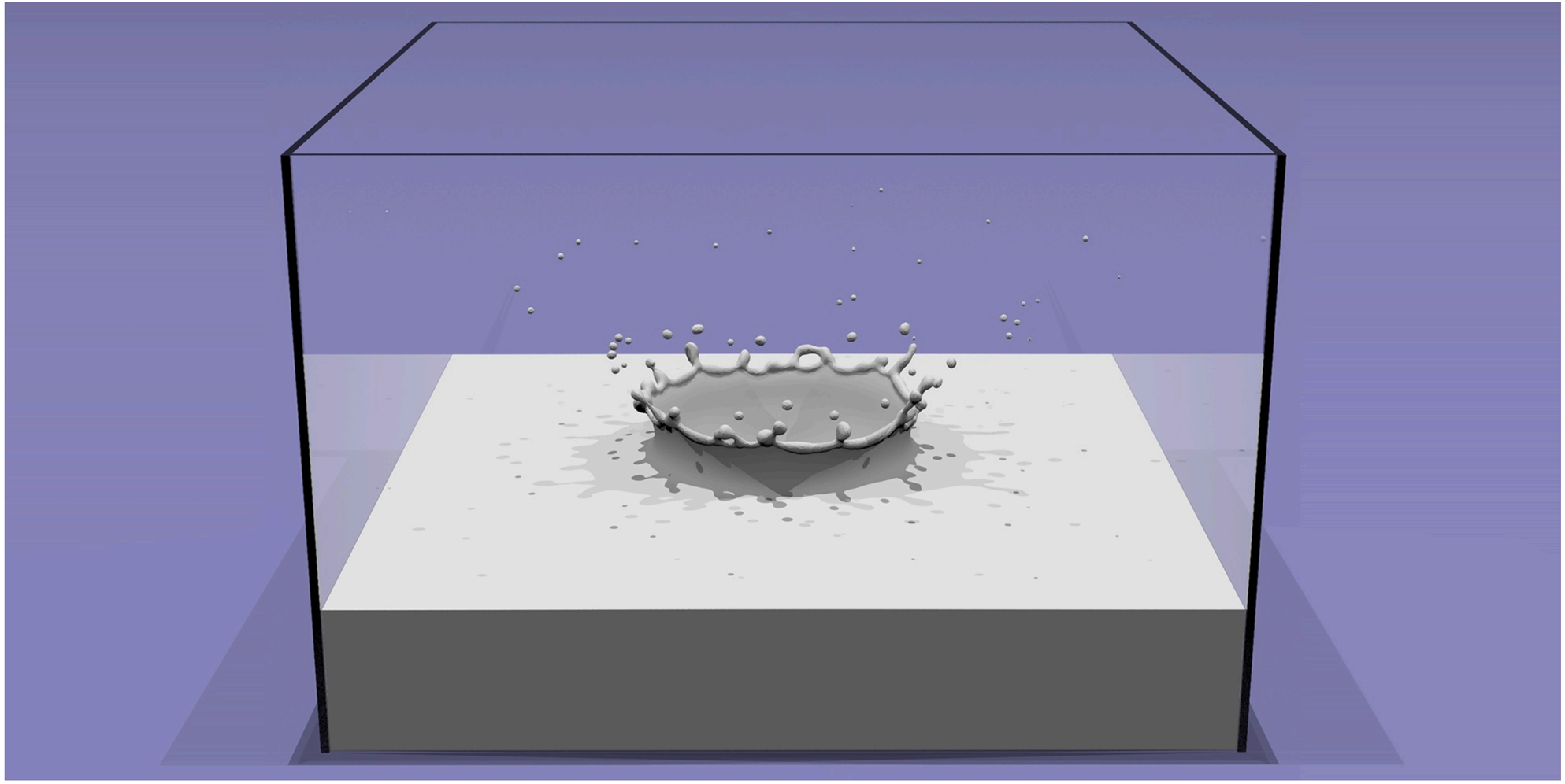
- Surface is determined by where the *interpolated* value equals zero

Example: sampled distance functions in medical data (CT, MRI, etc.)



Another example: tabulated distance functions in physical simulation

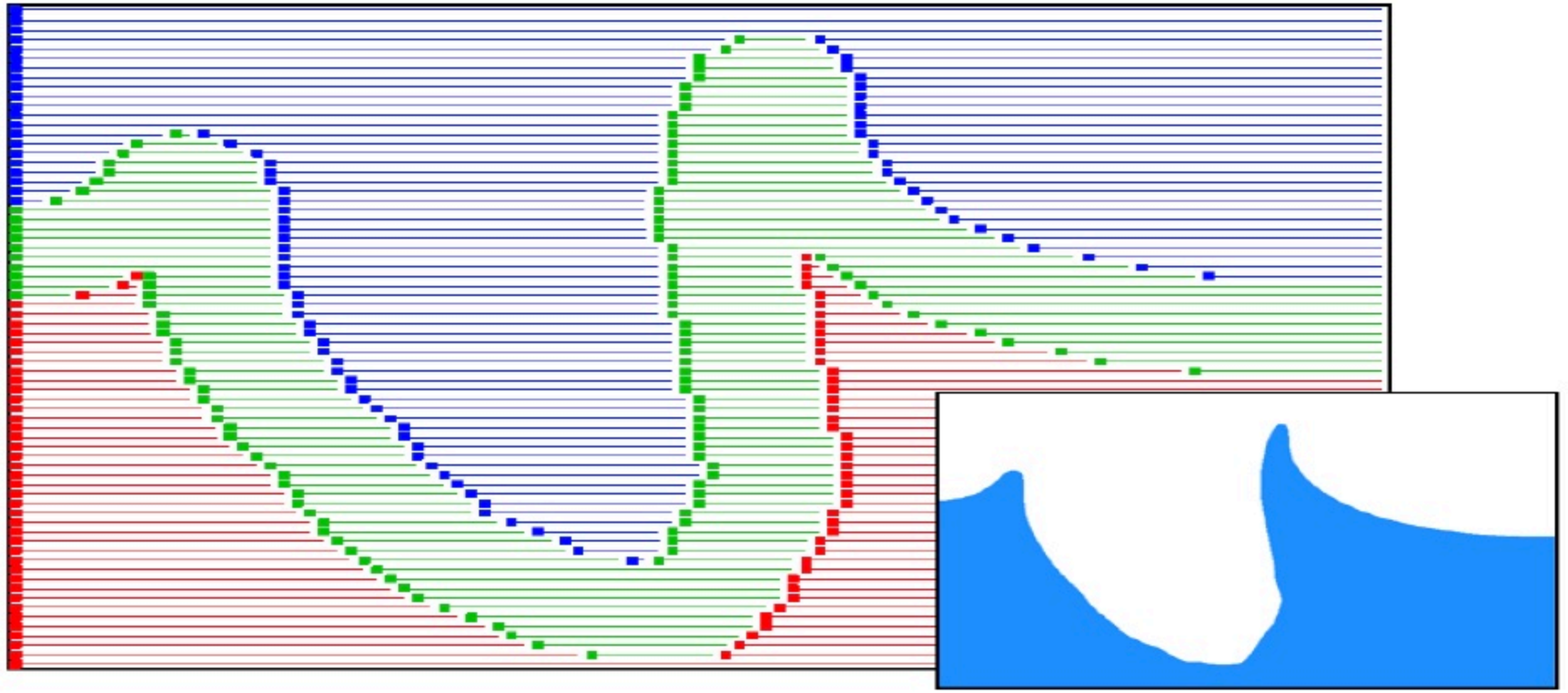
Encode distance to air-liquid boundary



See http://physbam.stanford.edu

Need for sparse storage representations

- Drawback: storage for surface is now $O(n^3)$
- Can reduce storage cost using sparse data structures that store only a narrow band of distances around surface (don't waste storage for empty space)
- But sparse structures can be difficult to implement efficiently on modern parallel computers



In this figure:
red = clearly within water
blue = clearly outside water

green = regions where we store level set values to encode surface

Occupancy field

Store a bit per cell in a dense 3D grid that indicates whether the cell center is inside the surface (3D array of inside/outside samples)

Consider storage requirements:

4096^3 cells, consider 1 bit/cell → ~ 8 GB

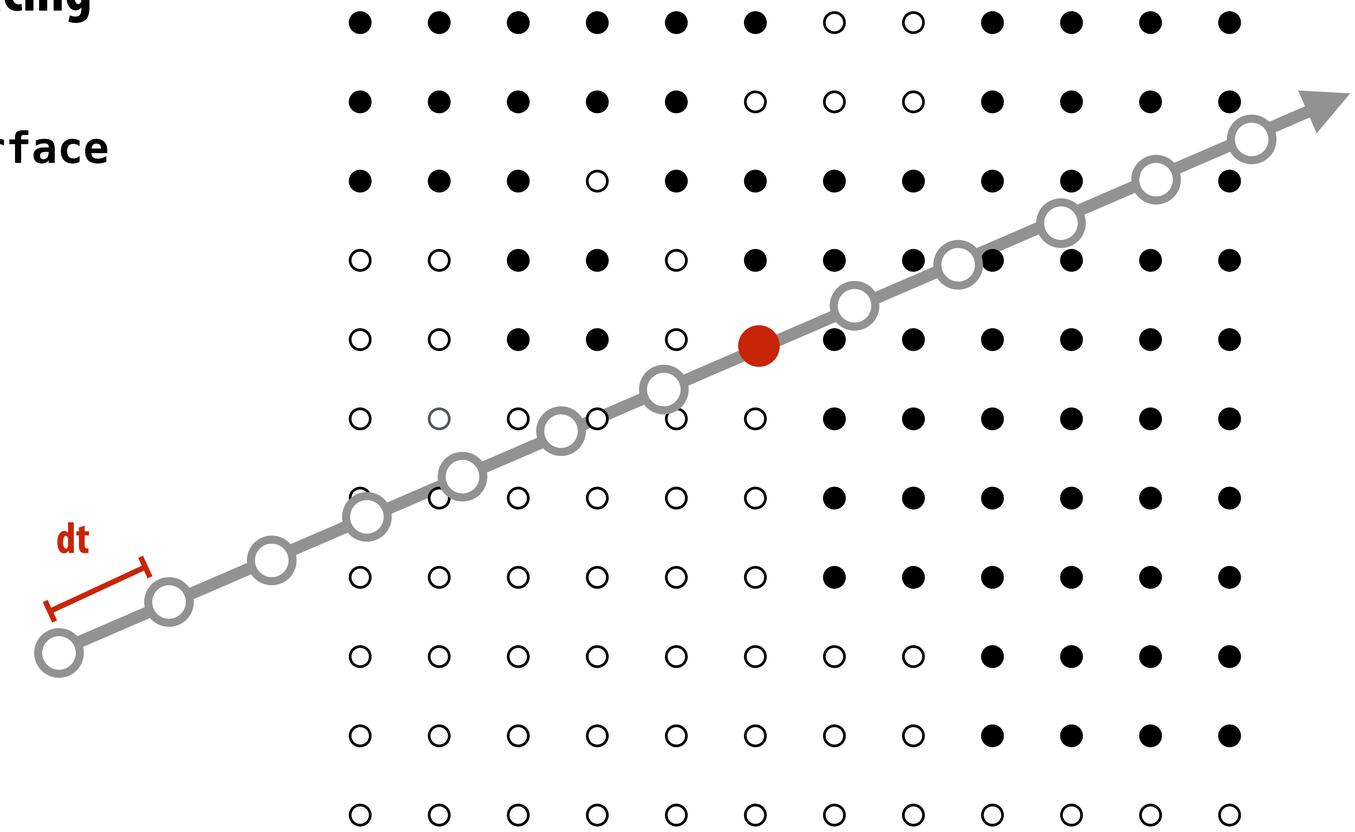
Typical challenge: limited resolution

Credit: Voxel Ville NFT (voxelville.io)

Example task: ray tracing occupancy grid

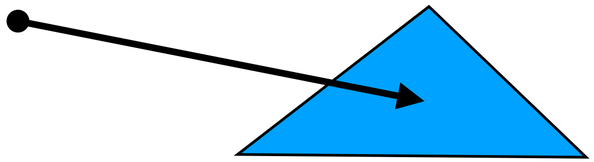
- Since we have no information about distance along ray to surface, we just have to "step" slowly until we find the surface
- Often called "ray marching" instead of ray tracing

```
# return hotpoint and distance to surface
for i = 0 to MAX_STEPS:
   cur_t = (i * dt)
   p = ray.o + ray.d * cur_t

   // interpolation of samples
   occupancy = sample_occupancy(p)

   if (occupancy > 0)
      return (p, cur_t)
```

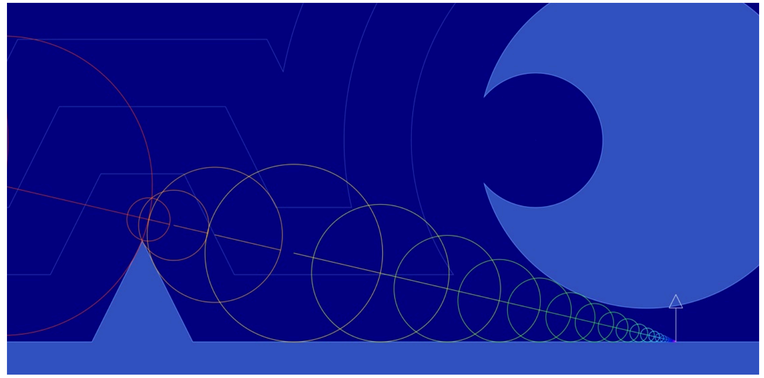


Comparing three representations for ray-surface intersections

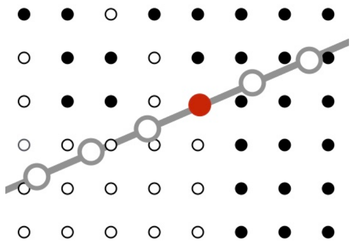
- **Ray-triangle:**
  - Can solve for distance from ray origin *along ray* to closest point on triangle



- **Ray-distance function**
  - Can determine closest distance from point to surface (in any direction), but not in the direction along the desired ray
  - So we must take variable sized "jumps" along the ray



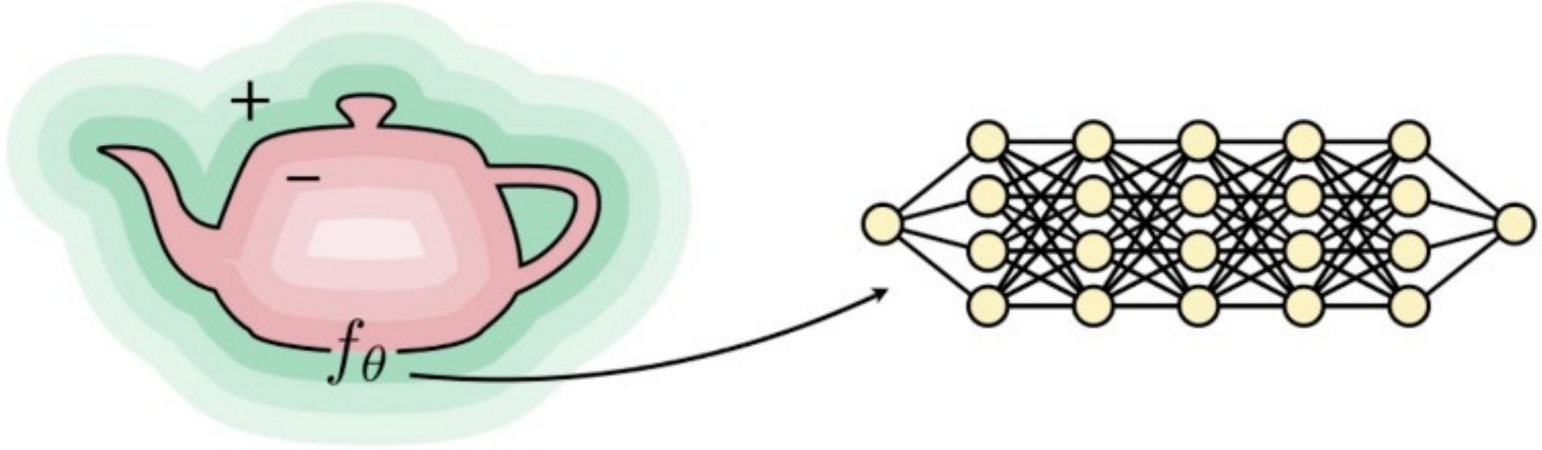
- **Ray-occupancy grid**
  - Don't know anything about the distance to the surface, so must march along in tiny steps *



* We'll talk about how adaptive data structures can accelerate this in a future lecture

Neural representations for compressing implicit representations of complex geometries

- Implicit forms boil down to having a function f(x,y,z)
- f(x,y,z) = c is the surface point (often we use c=0)
- Neural networks are function approximations…
- So just train a `neural_network(x,y,z)` using an existing function f(x,y,z) as training data! (e.g., convert SDF representations we're talked about to a NN using supervised learning!



Neural representations for compressing implicit representations of complex geometries

- **Simple solution:**
  - Train a DNN to evaluate f(x,y,z)
  - e.g., use conventional dense grid representation to create training data pairs
  - Good: massive compression (surface represented by weights of DNN, not a bunch of 3D occupancy grid or SDF samples)
  - Bad: high evaluation cost (must evaluate large DNN to determining distance from surface, instead of interpolate samples or evaluate a polynomial!
- **In practice: most modern approaches are "hybrid" approaches (ask me for more details)**
  - Use neural code to represent local surface structure
  - Store neural "code" at cells of traditional uniform grid, or sparse grid
    - e.g., code[x,y,z]
  - Train a "tiny" DNN to produce f(x,y,z) = DNN(x,y,z, code[x,y,z])
  - Idea: DNN only has to translate code into a function value = much cheaper to evaluate

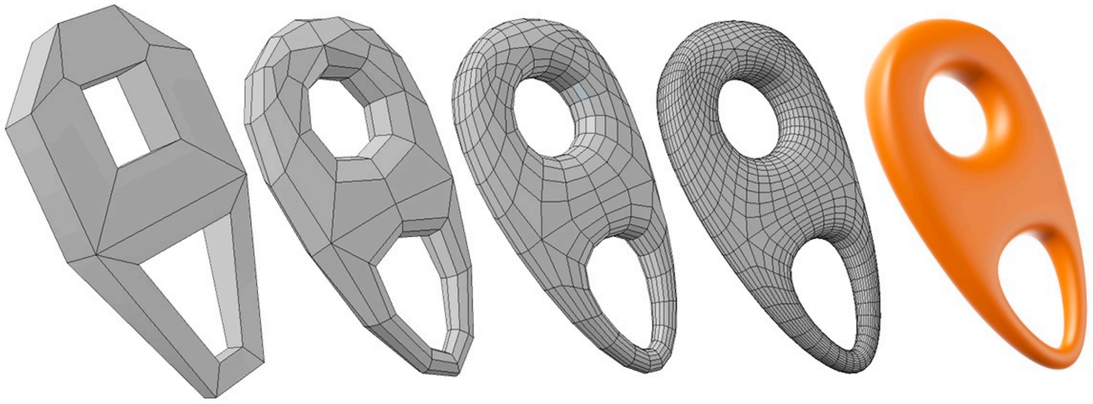
Implicit representations - pros and cons

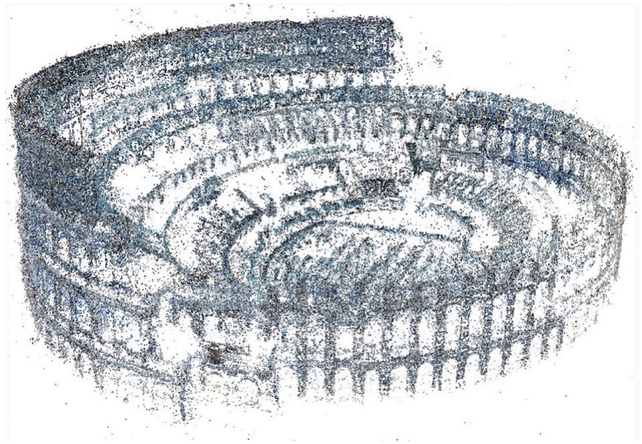
- **Pros:**
  - Description can be very compact (e.g., a polynomial, a neural network!)
  - Easy to determine if a point is in our shape (just plug it in!)
  - Other queries may also be easy (e.g., distance to surface)
  - For simple shapes, can provide an exact description/no sampling error
  - Easy to handle changes in topology (e.g., fluid)

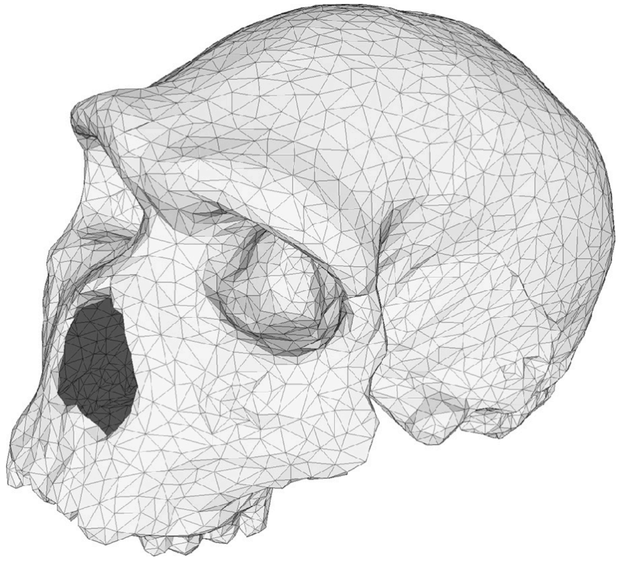
- **Cons:**
  - Expensive to find all points in the shape (e.g., for drawing)
  - *Traditionally it has difficult to model complex shapes, but efficient sparse data structures (for sampled representations) or learned “neural" representations change this*

Also many explicit representations in graphics

- triangle meshes
- polygon meshes
- subdivision surfaces
- point clouds
- 3D gaussians



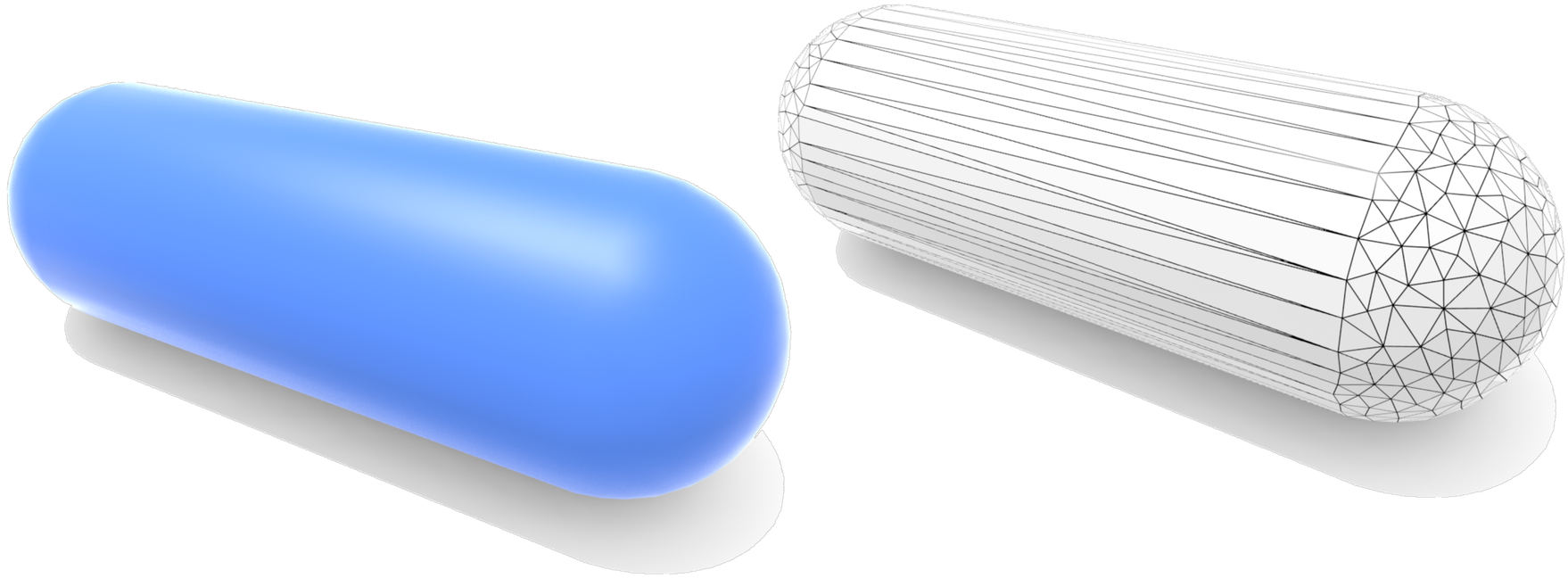




(Will see some of these a bit later.)

Polygon mesh (explicit)

- Store vertices *and* polygons (most often triangles or quads)
- Perhaps most common representation in graphics



Triangle mesh (explicit)

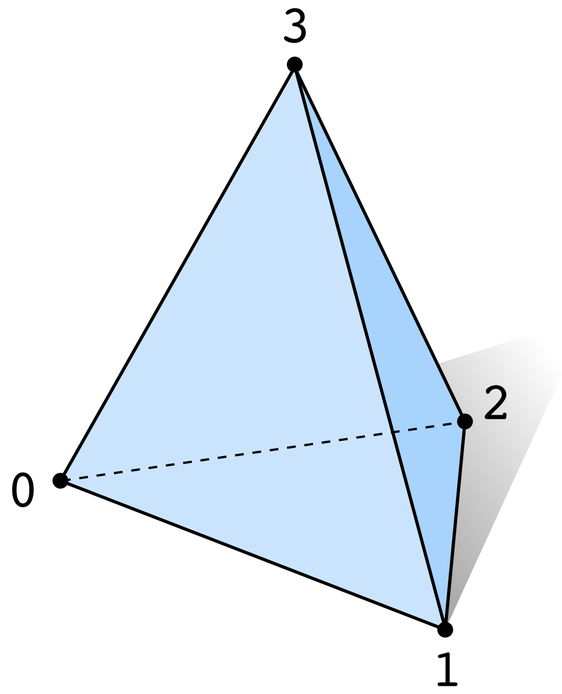
- Store vertices as triples of coordinates (x,y,z)
- Store triangles as triples of indices (i,j,k)
- E.g., tetrahedron:

VERTICES

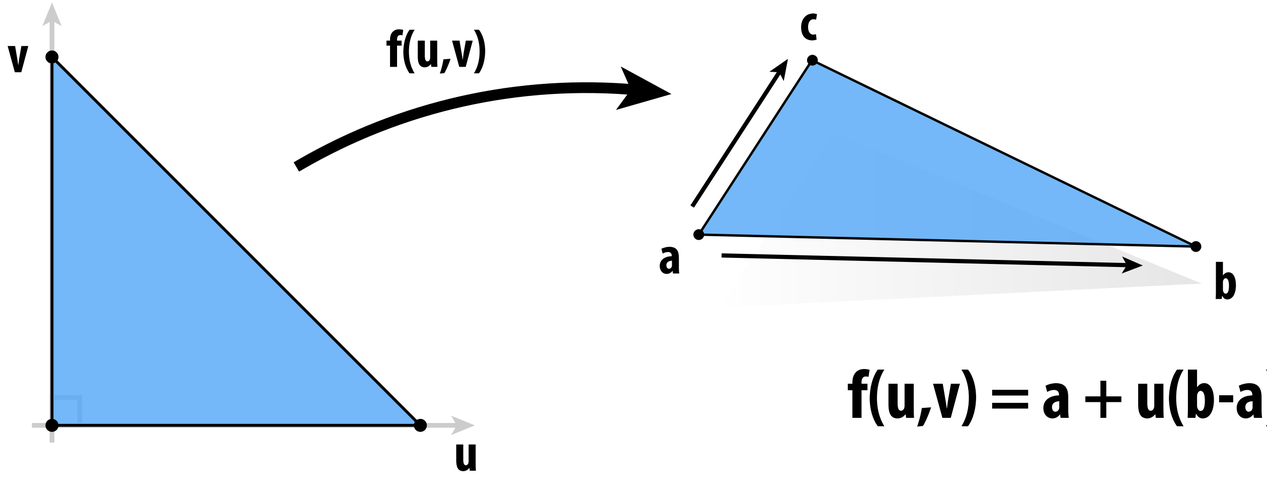
	x	y	z
0:	-1	-1	-1
1:	1	-1	1
2:	1	1	-1
3:	-1	1	1

TRIANGLES

	i	j	k
	0	2	1
	0	3	2
	3	0	1
	3	1	2



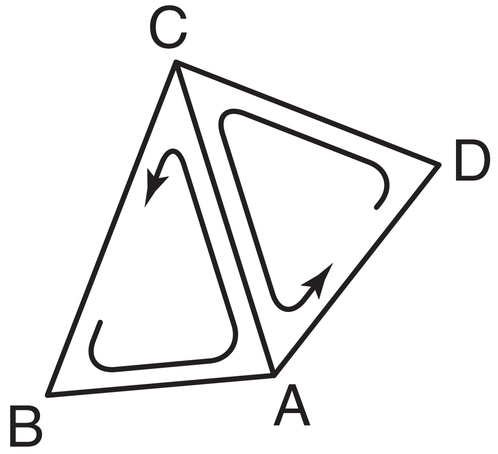
- Recall from earlier in the lecture: use linear interpolation to define points inside triangles:



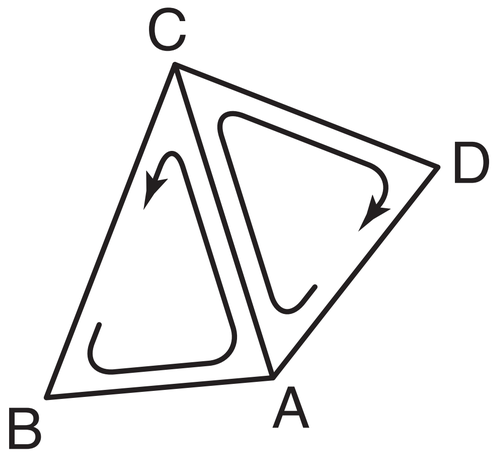
$$f(u,v) = a + u(b-a) + v(c-a)$$

Topological validity: orientation consistency

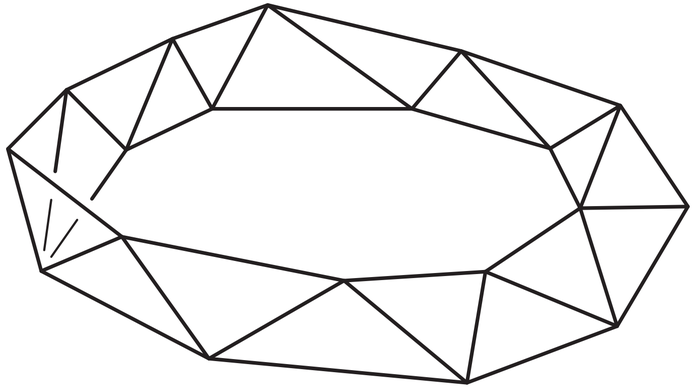
Both facing front



Inconsistent orientations



Non-orientable
(e.g., Moebius strip)



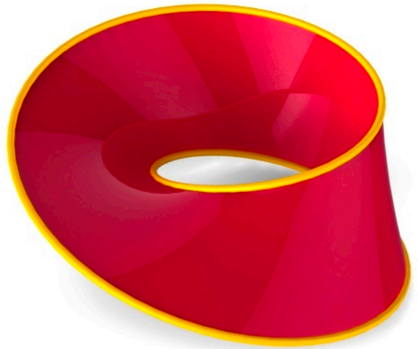
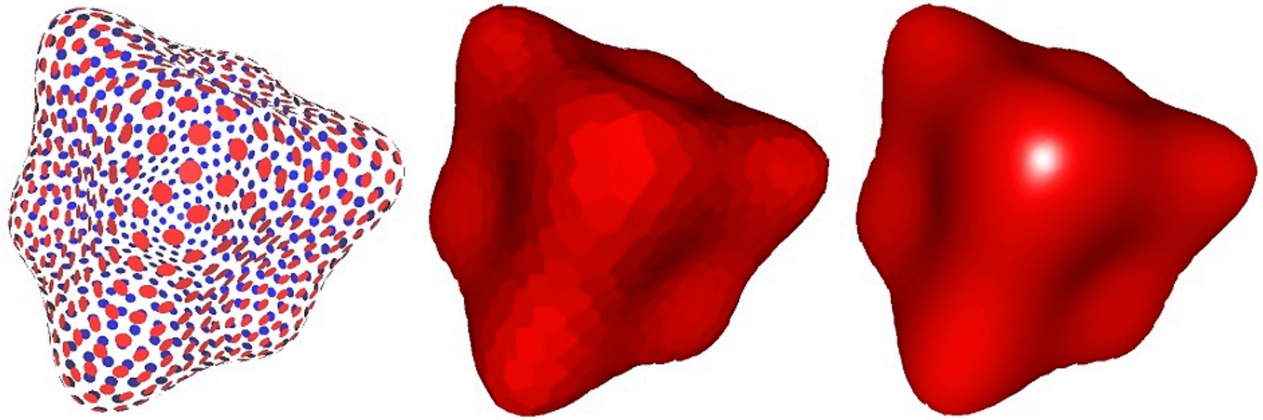
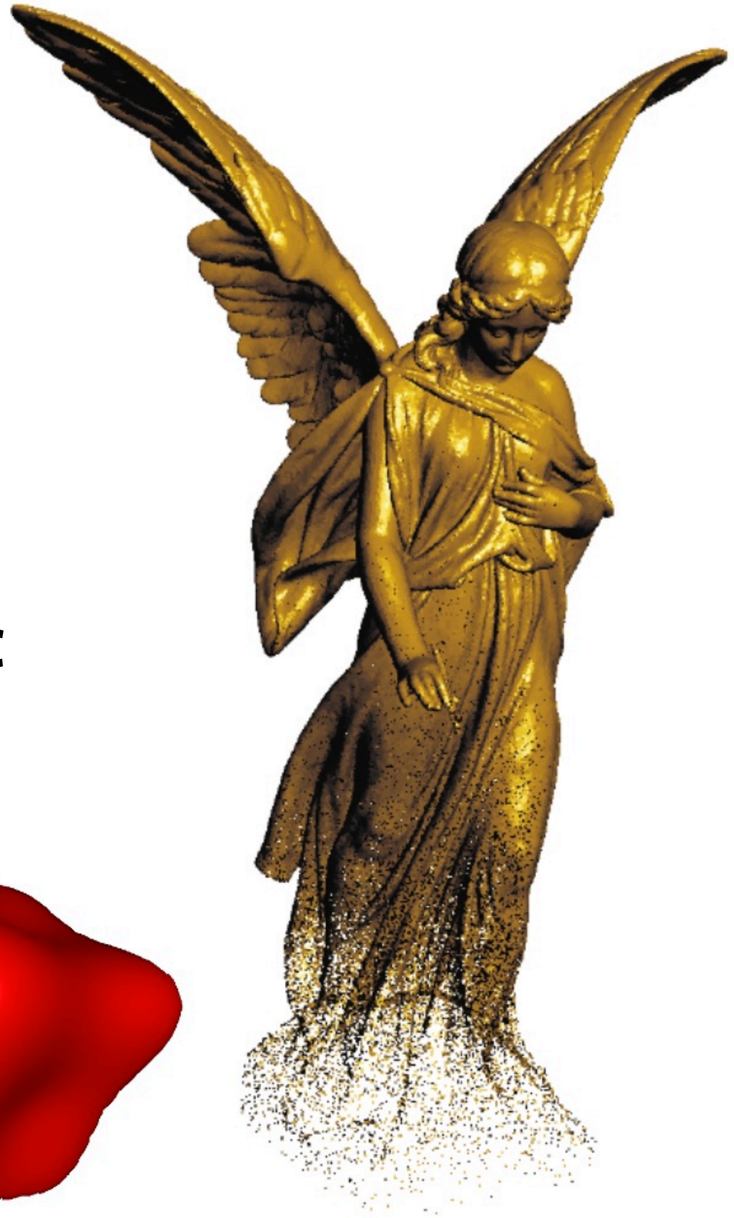


Image credit: Wikipedia

Point cloud (explicit)

- List of points (x,y,z)
- Often augmented with per-point *normals*
- Hard to interpolate undersampled regions
- Easier to acquire (laser scanner)
- Often challenging to do processing/simulation, etc
  … on this representation





Acquiring a point cloud via laser scanning

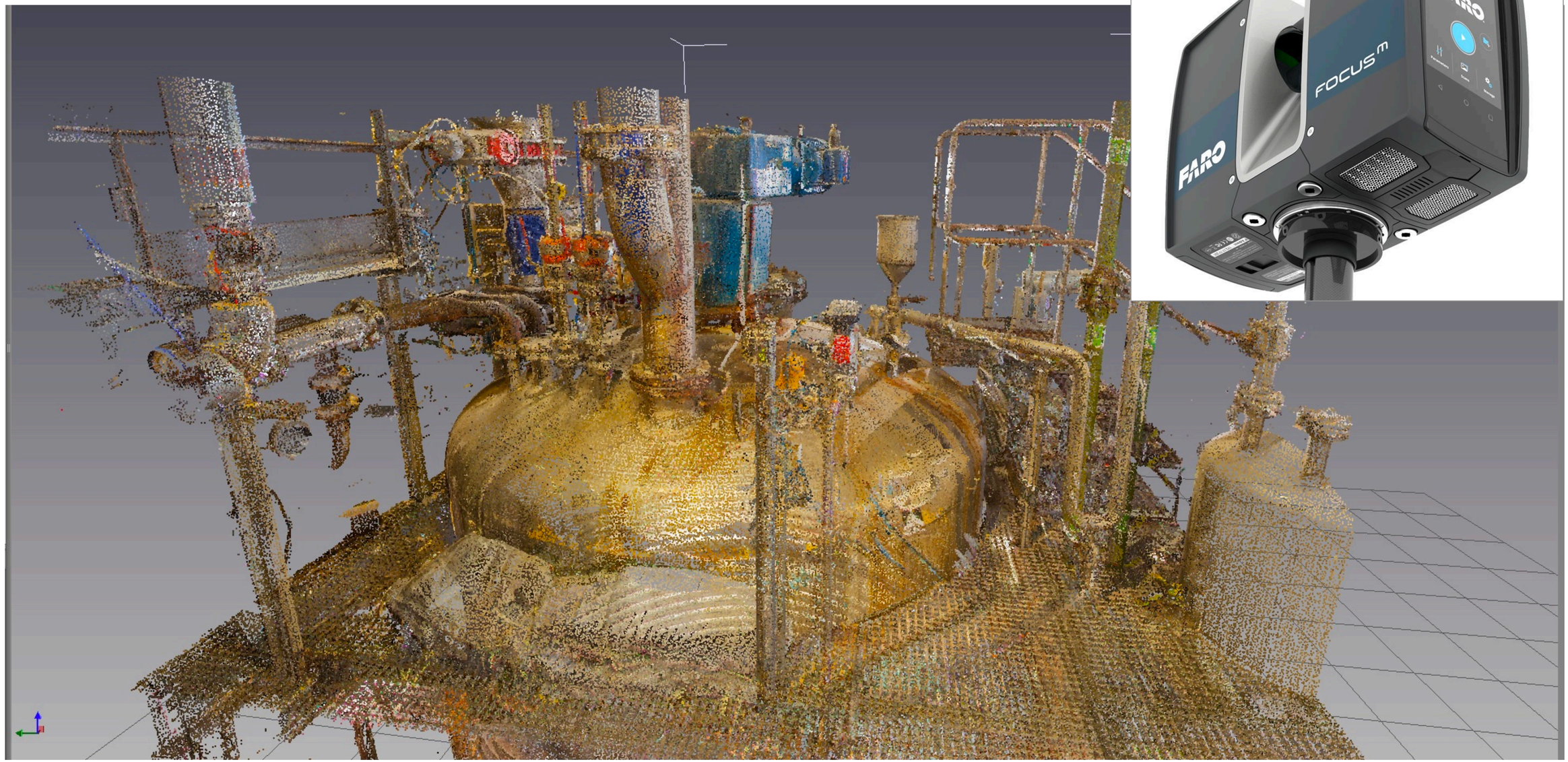


Image Credit: 3Dling

Another point acquisition example: Microsoft XBox 360 Kinect

Illuminant
(Infrared Laser + diffuser)

RGB Sensor
640x480

Monochrome Infrared
Sensor

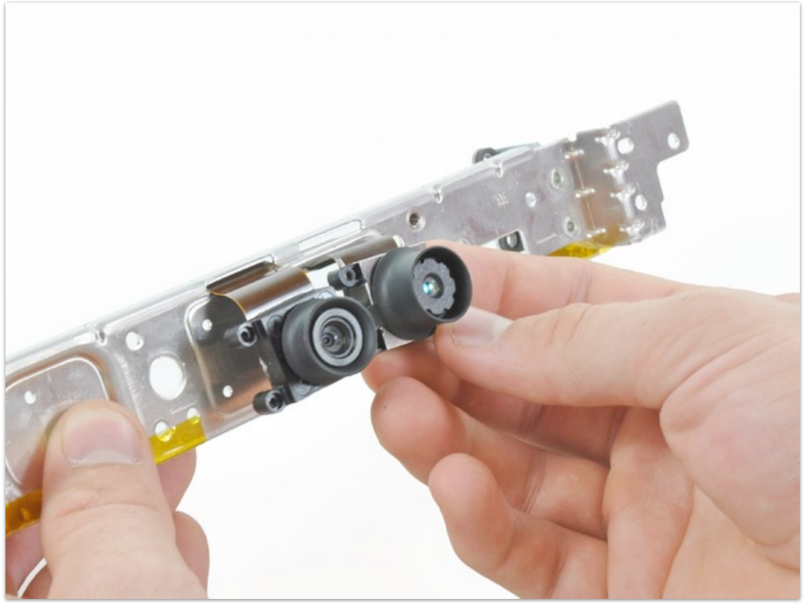
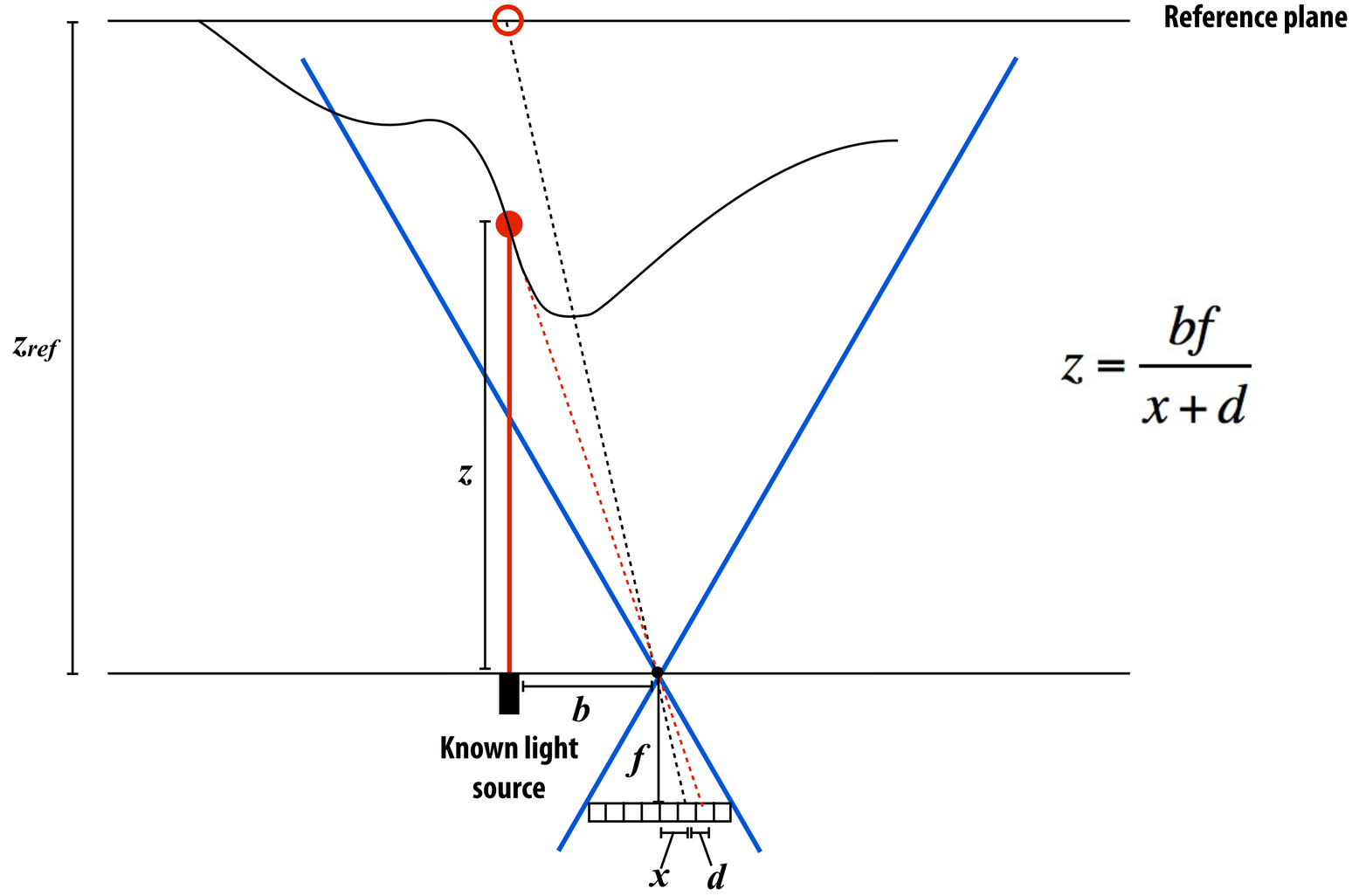


Image credit: iFixIt

Structured light

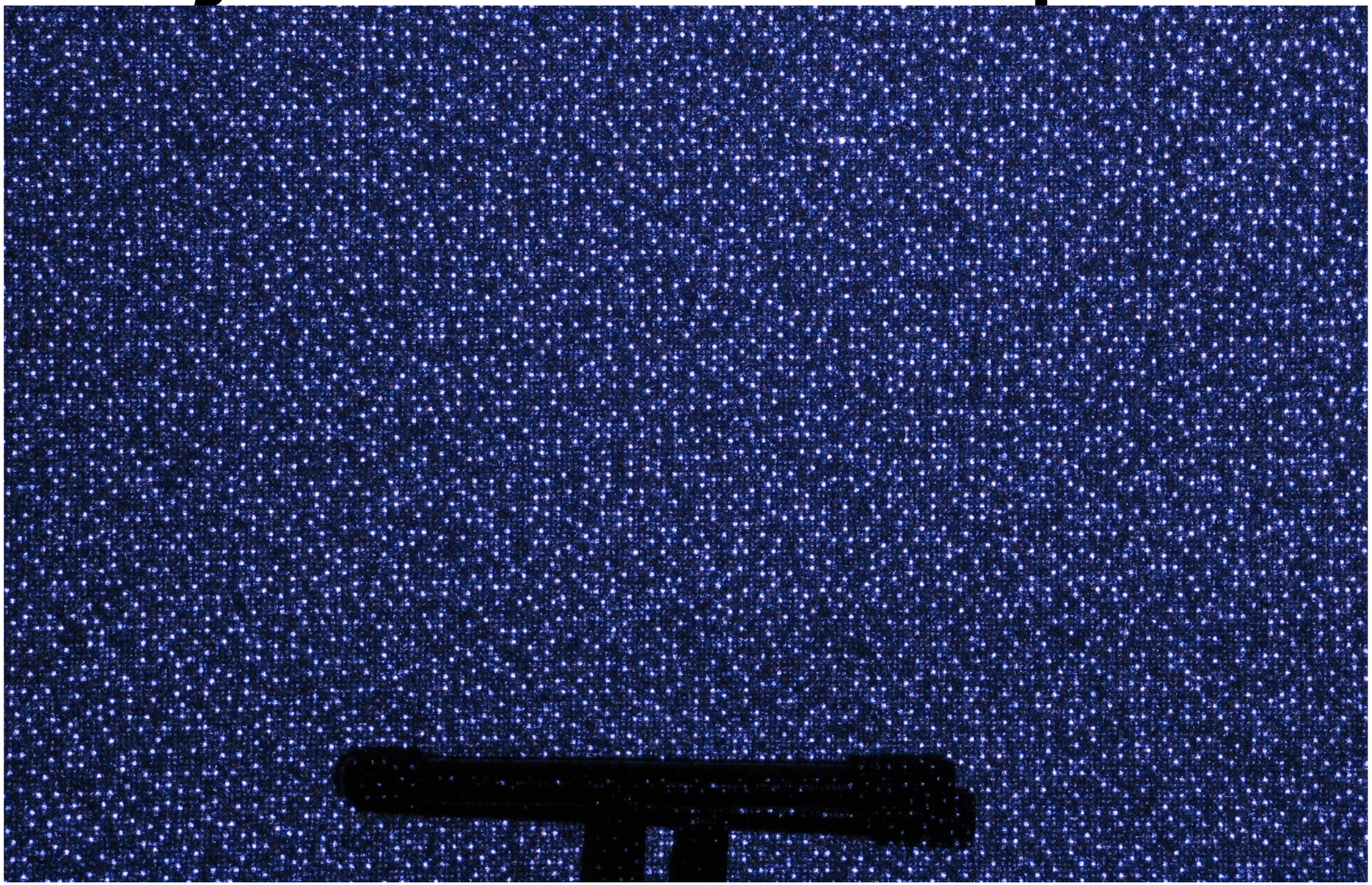
System: one light source emitting known beam + one camera measuring scene appearance
If the scene is at reference plane, image that will be recorded by camera is known
(correspondence between pixel in recorded image and scene point is known)



$$z = \frac{bf}{x + d}$$

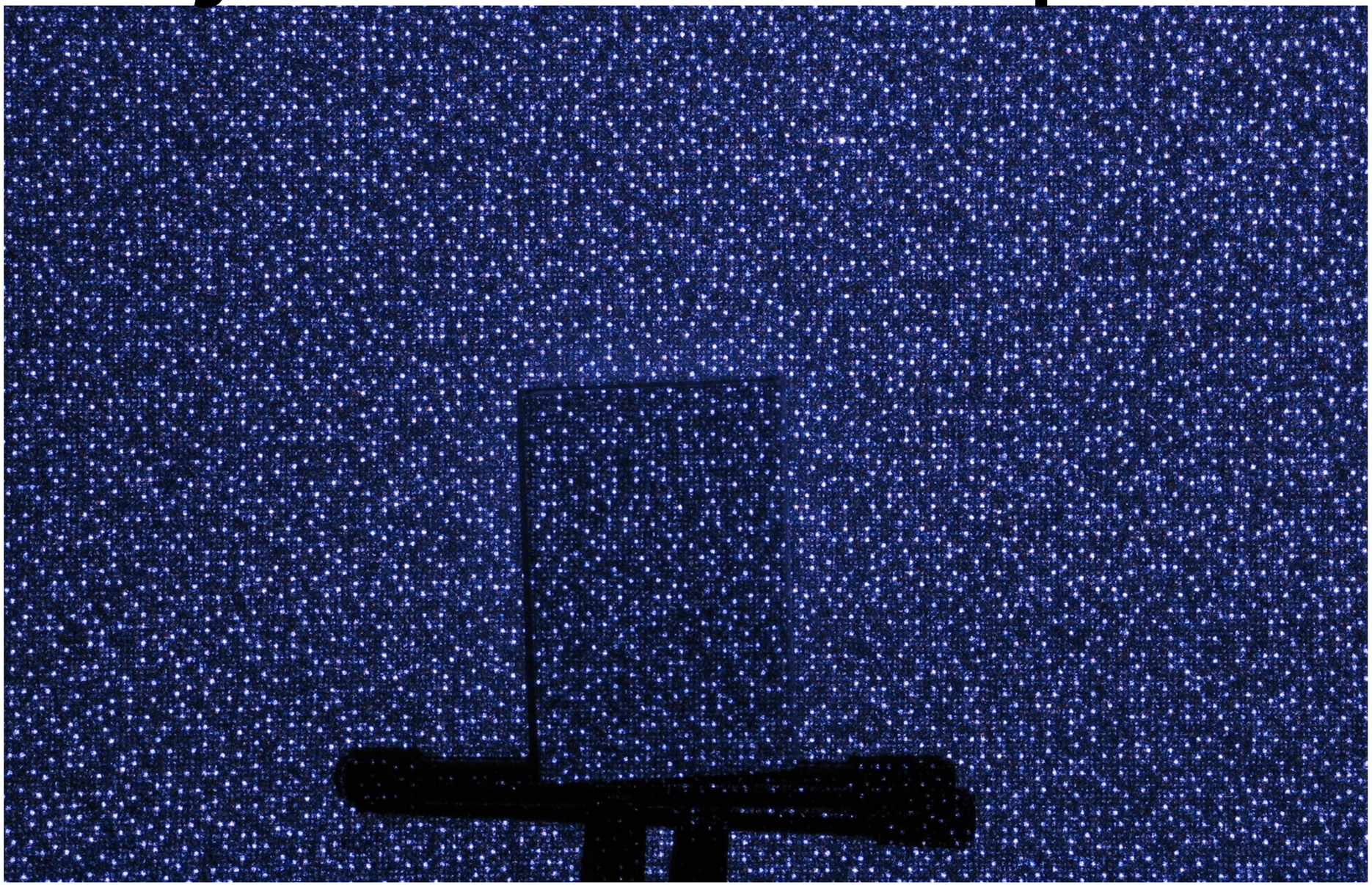
Single spot illuminant is inefficient!
(must "scan" scene with spot to get depth, so high latency to retrieve a single depth image)

Infrared image of Kinect illuminant output



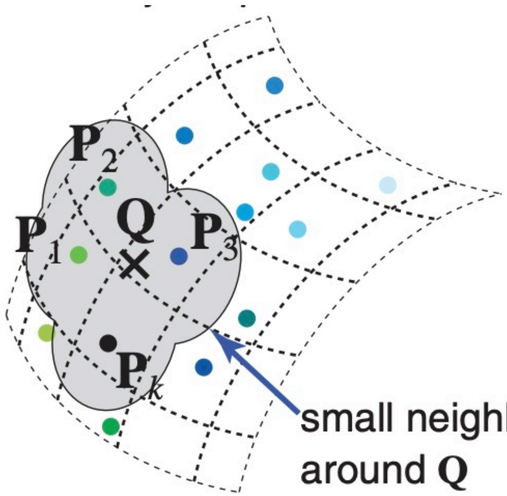
Credit: www.futurepicture.org

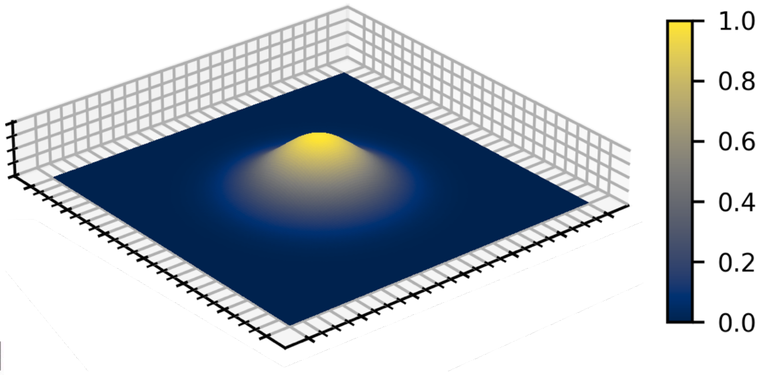
Infrared image of Kinect illuminant output



Credit: www.futurepicture.org

Oriented 3D Gaussians





Gaussian density kernel

- Instead of a point, represent surface as an oriented Gaussian ellipsoid around a point
- Each 3D Gaussian is:
  - Center point
  - Extents of ellipse in X,Y,Z direction (in ellipse's object space)
  - Rotation into world space
- Immensely popular representative for photoreal scenes in recent years
- Common questions: given a scene, how many Gaussians to use? (Many small Gaussians? Fewer large Gaussians?)

[Zwicker 2001]

Visualization of 3D ellipsoid centers

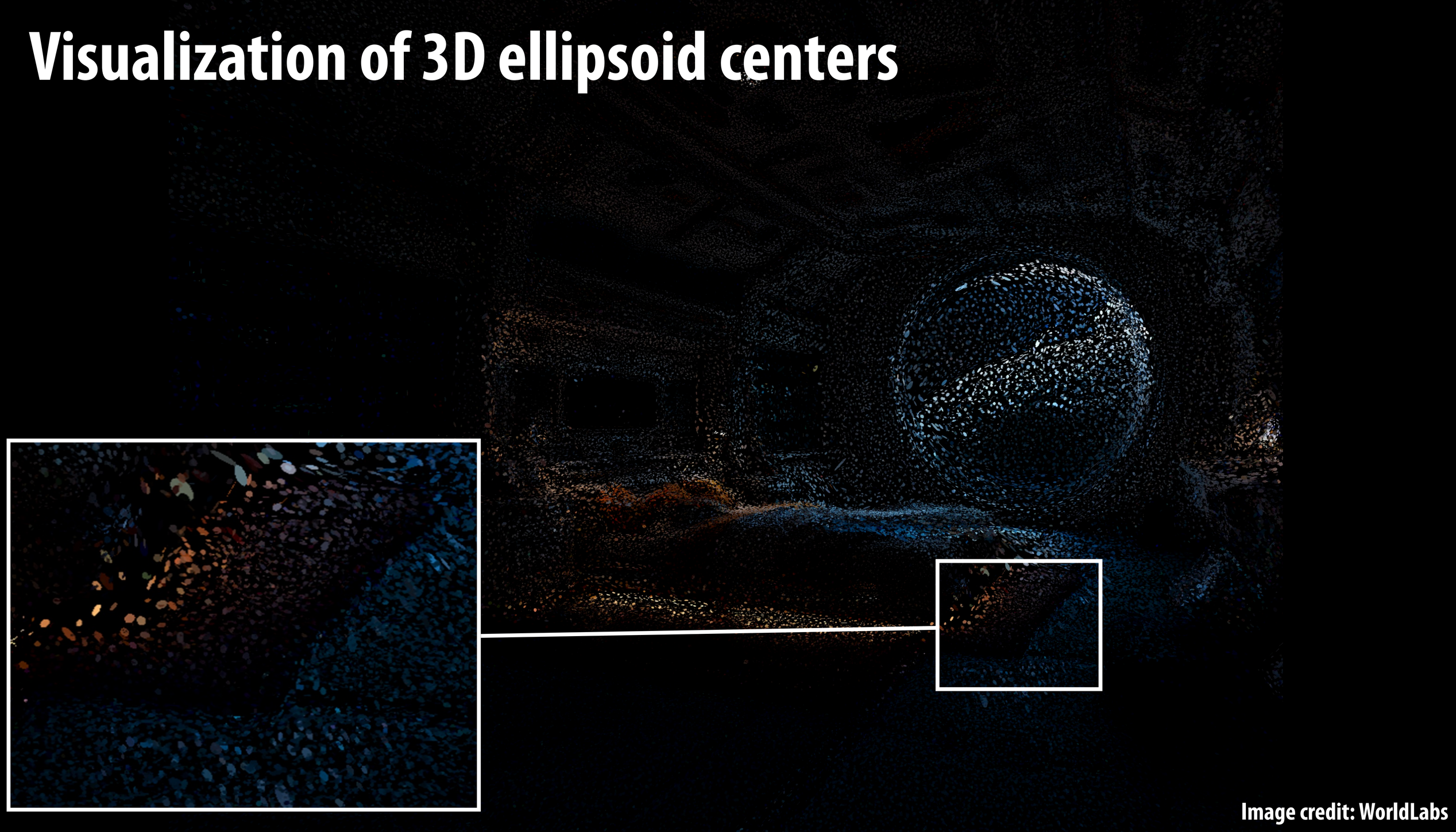


Image credit: WorldLabs

Visualization of oriented 3D ellipsoids

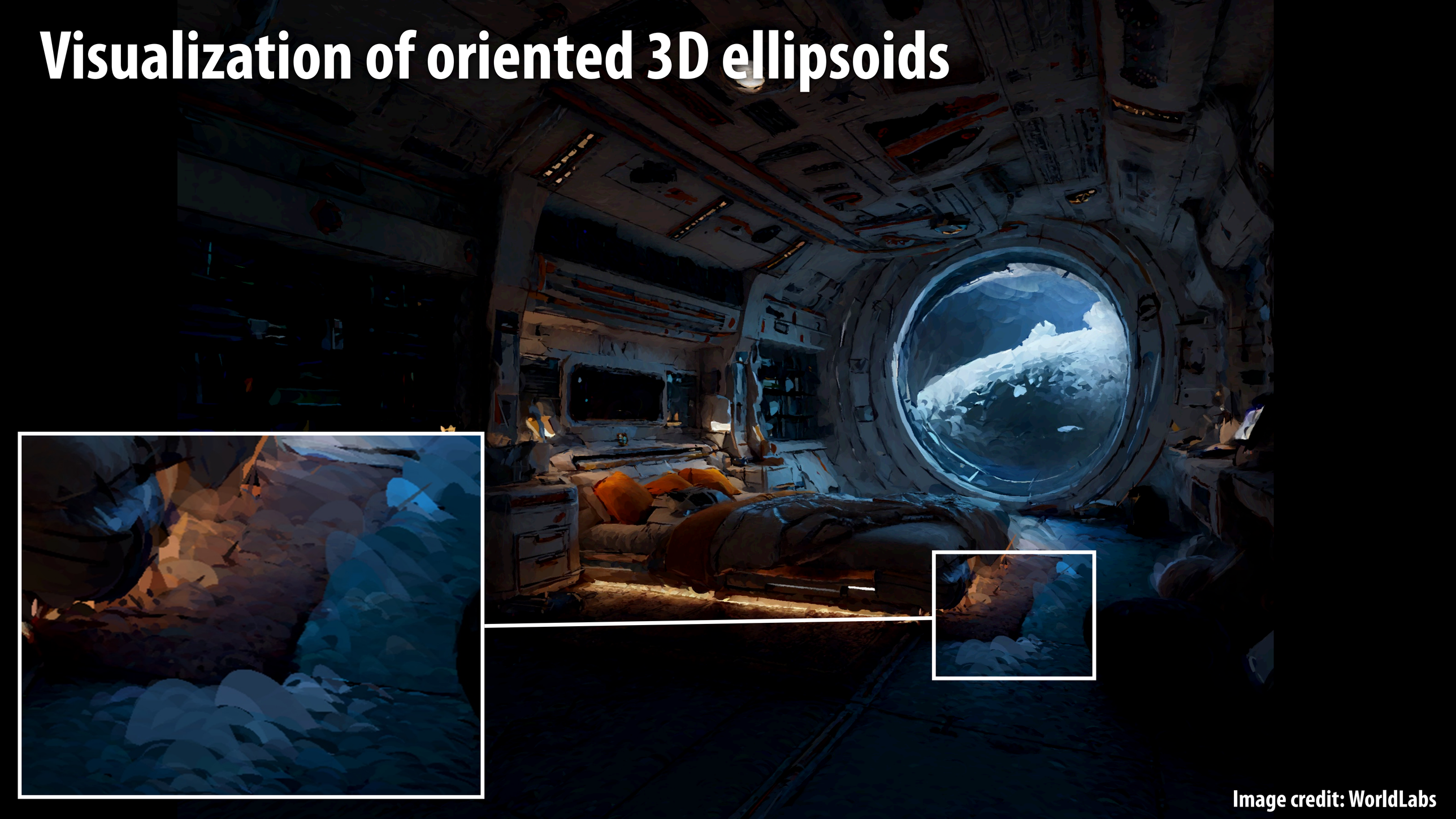


Image credit: WorldLabs

Rendered result

(oriented ellipsoids treated as densities given by Gaussian kernel)

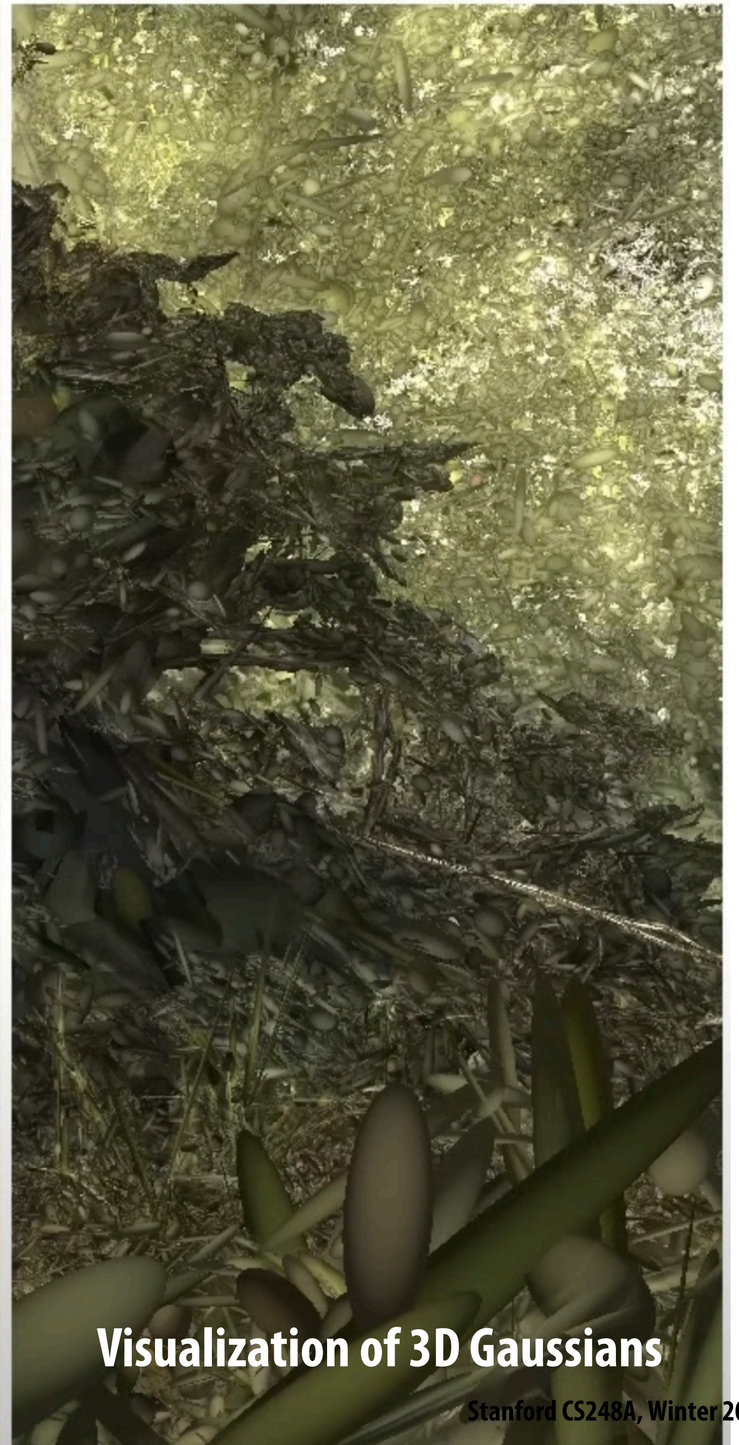
Image credit: WorldLabs

Rendered video

Image credit: WorldLabs

Visualization of oriented 3D Gaussians

Rendered Result



Visualization of 3D Gaussians

[Credit: Kerbl 2023]

Rendering of scene represented as 3D gaussians

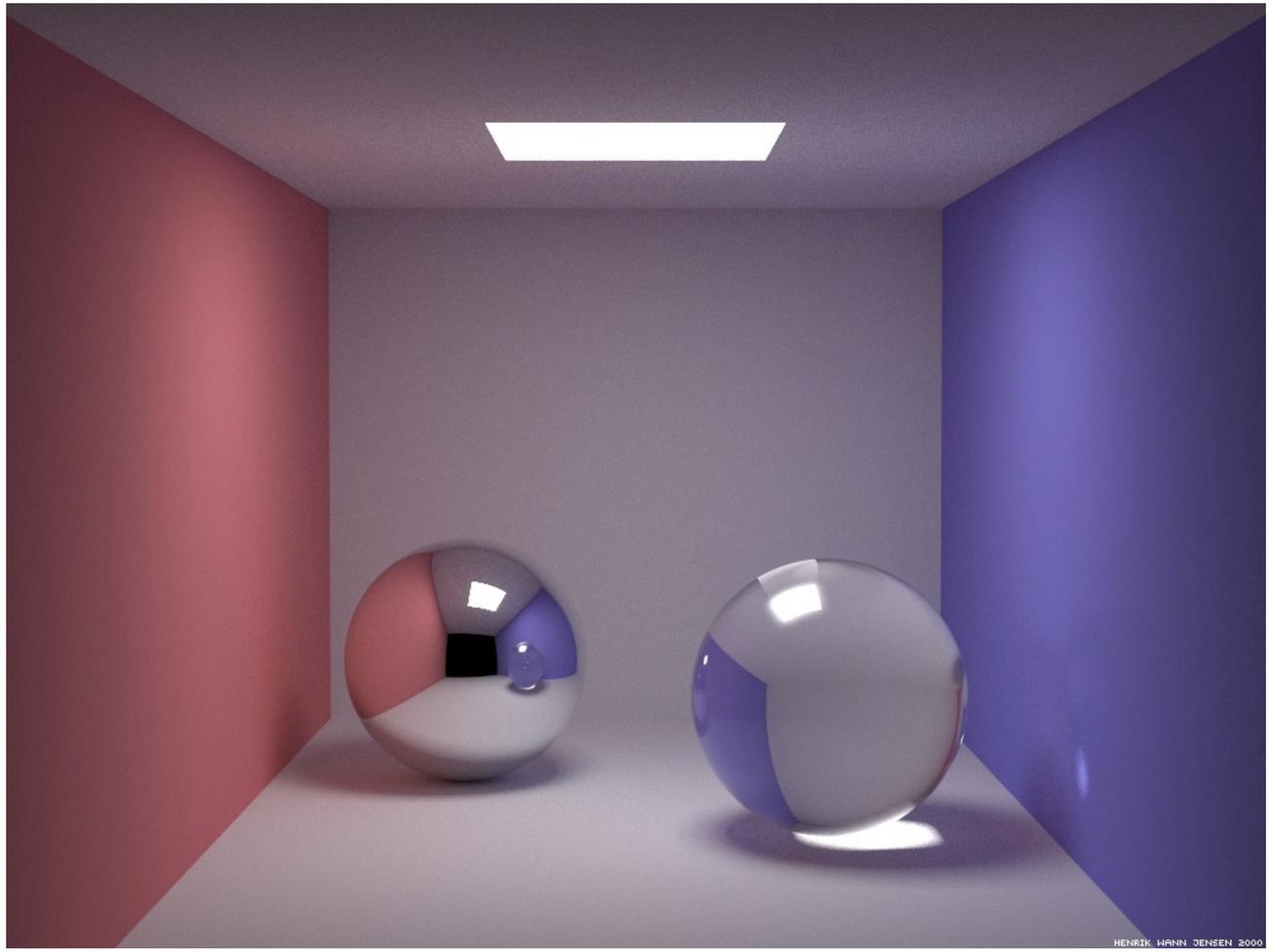
[Credit: Kerbl 2023]

Some questions for the class

If you tell me a task, and then we can access the utility of different representations

- **Describe the characteristics of the scene that needs to be represented**
- **Describe what operation you want to perform on the scene geometry:**
  - **Rendering/visualization?**
  - **Animation?**
  - **Editing?**
  - **Reducing detail?**
  - **Finding the closest scene surface to a given point?**
  - **Estimating the surface normal?**
  - **Recovering parameters to fit a photo?**

For example:
Consider representing this scene with ten's of thousands of gaussians vs. two spheres and a few triangles

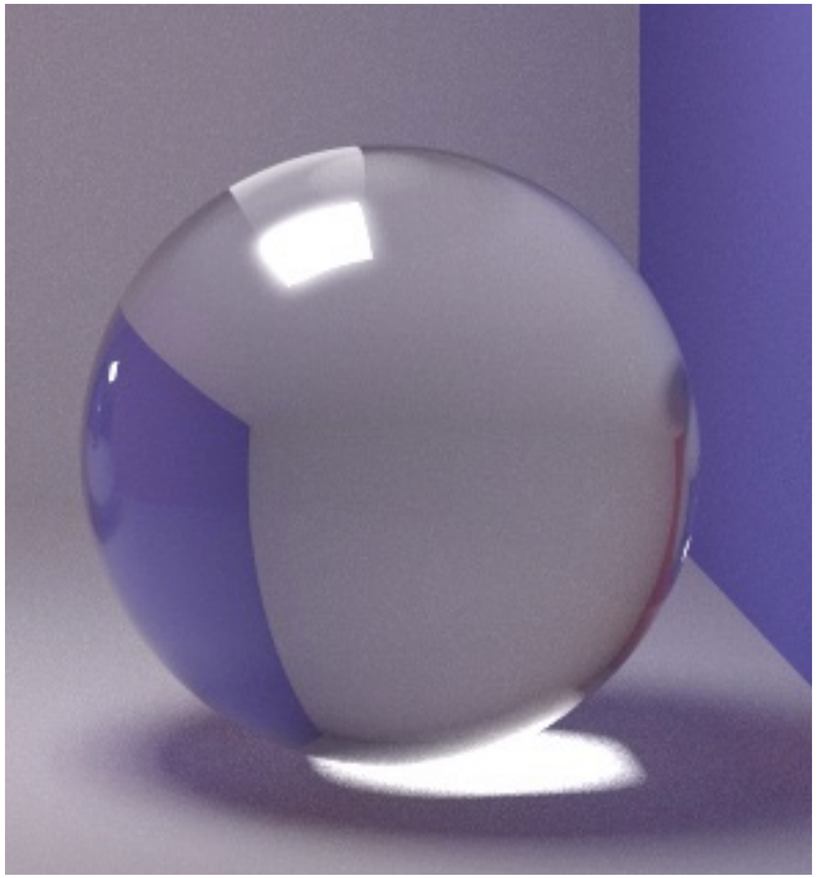


Some questions for the class

If you tell me a task, and then we can access the utility of different representations

- **Describe the characteristics of the scene that needs to be represented**
- **Describe what operation you want to perform on the scene geometry:**
  - **Rendering/visualization?**
  - **Animation?**
  - **Editing?**
  - **Reducing detail?**
  - **Finding the closest scene surface to a given point?**
  - **Estimating the surface normal?**
  - **Recovering parameters to fit a photo?**

Does it make sense to represent this curved surface with a voxel grid? How many voxels would you need?



Some questions for the class

If you tell me a task, and then we can access the utility of different representations

- **Describe the characteristics of the scene that needs to be represented**
- **Describe what operation you want to perform on the scene geometry:**
  - **Rendering/visualization?**
  - **Animation?**
  - **Editing?**
  - **Reducing detail?**
  - **Finding the closest scene surface to a given point?**
  - **Estimating the surface normal?**
  - **Recovering parameters to fit a photo?**

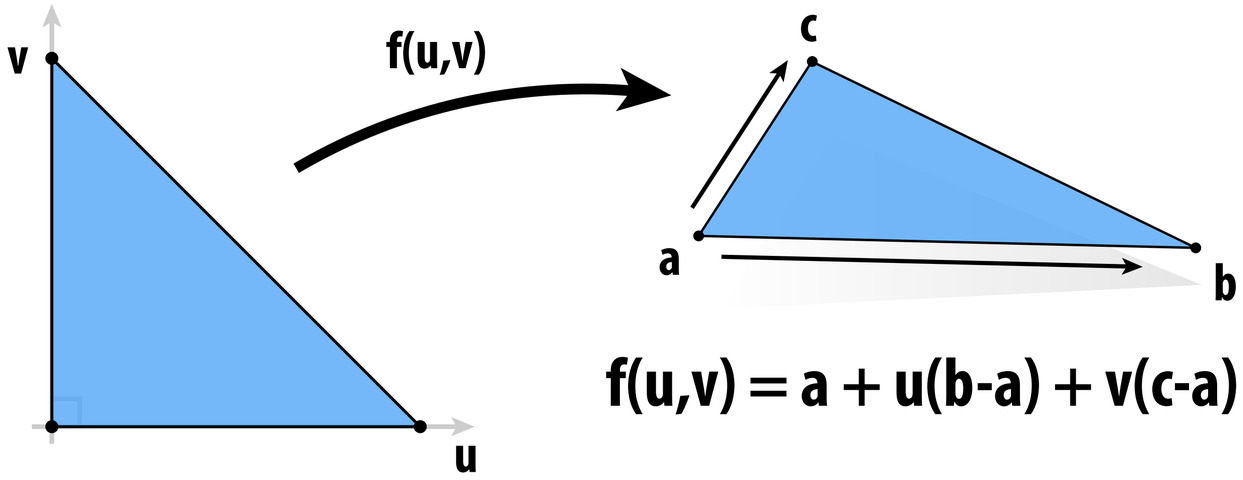
But what about accurately representing these scenes with triangles?

Moving from linear interpolation to higher order interpolation of surface points

(First… back to triangles)

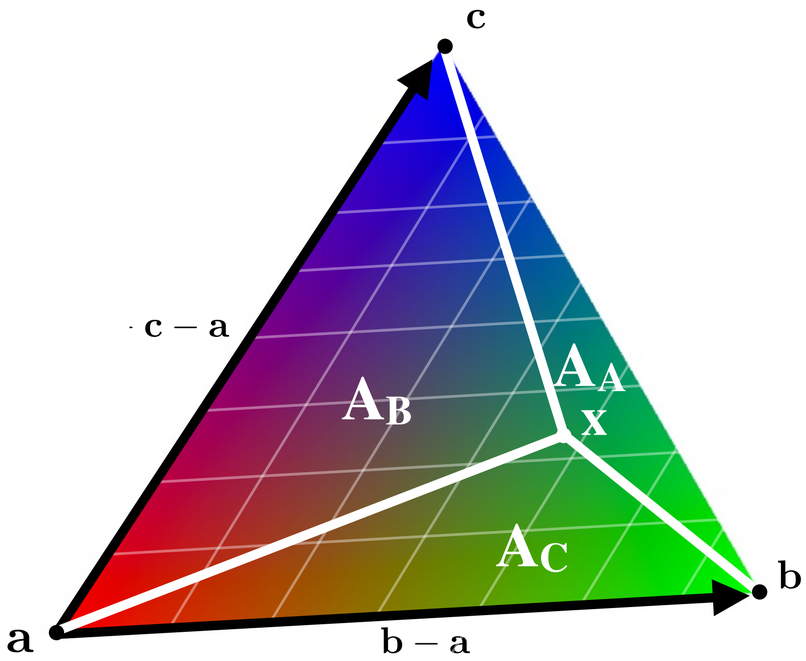
Points in a triangle = linear interpolation of vertices

Recall from earlier in the lecture: use linear interpolation to define points inside triangles:



$$f(u,v) = a + u(b-a) + v(c-a)$$

Specifically barycentric interpolation as a form of 2D interpolation

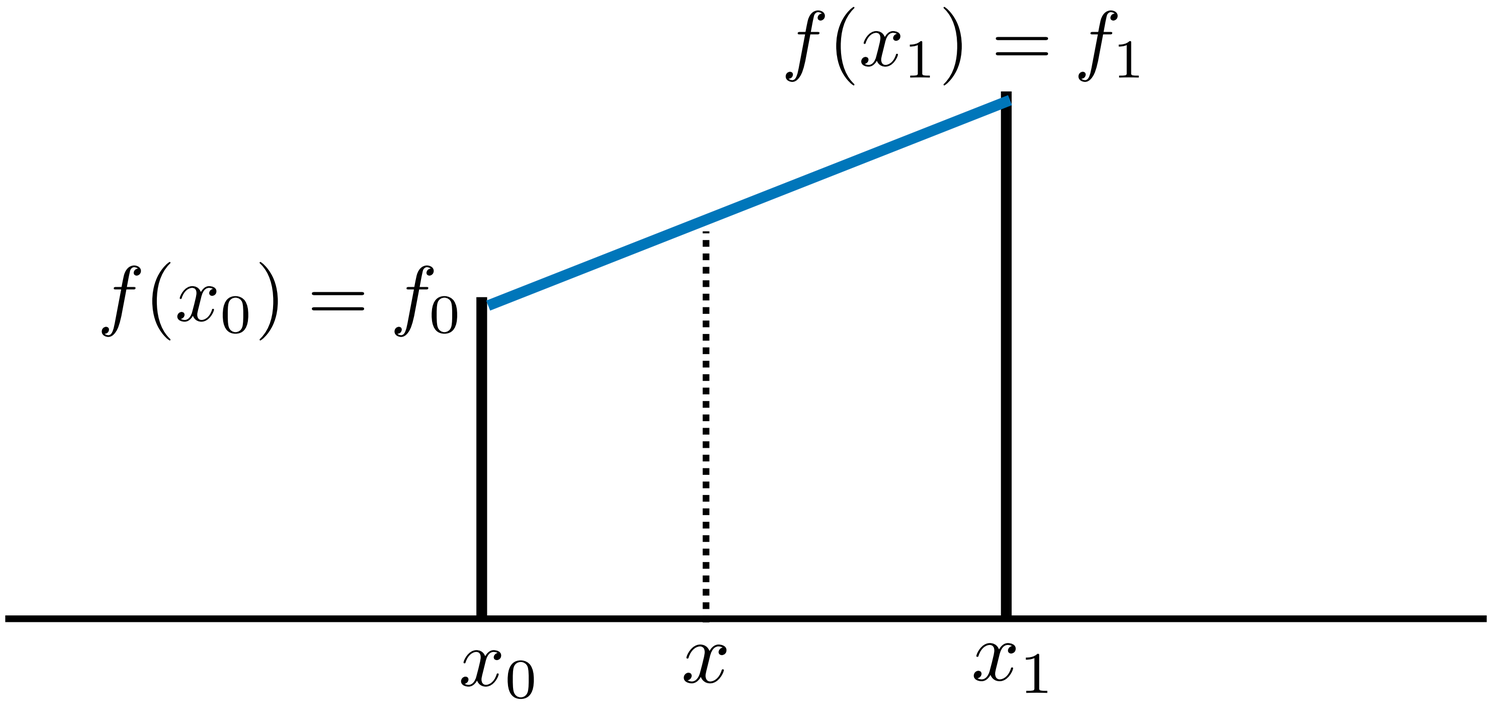


$$\alpha = A_A/A$$

$$\beta = A_B/A$$

$$\gamma = A_C/A$$

Linear interpolation of samples (in 1D)

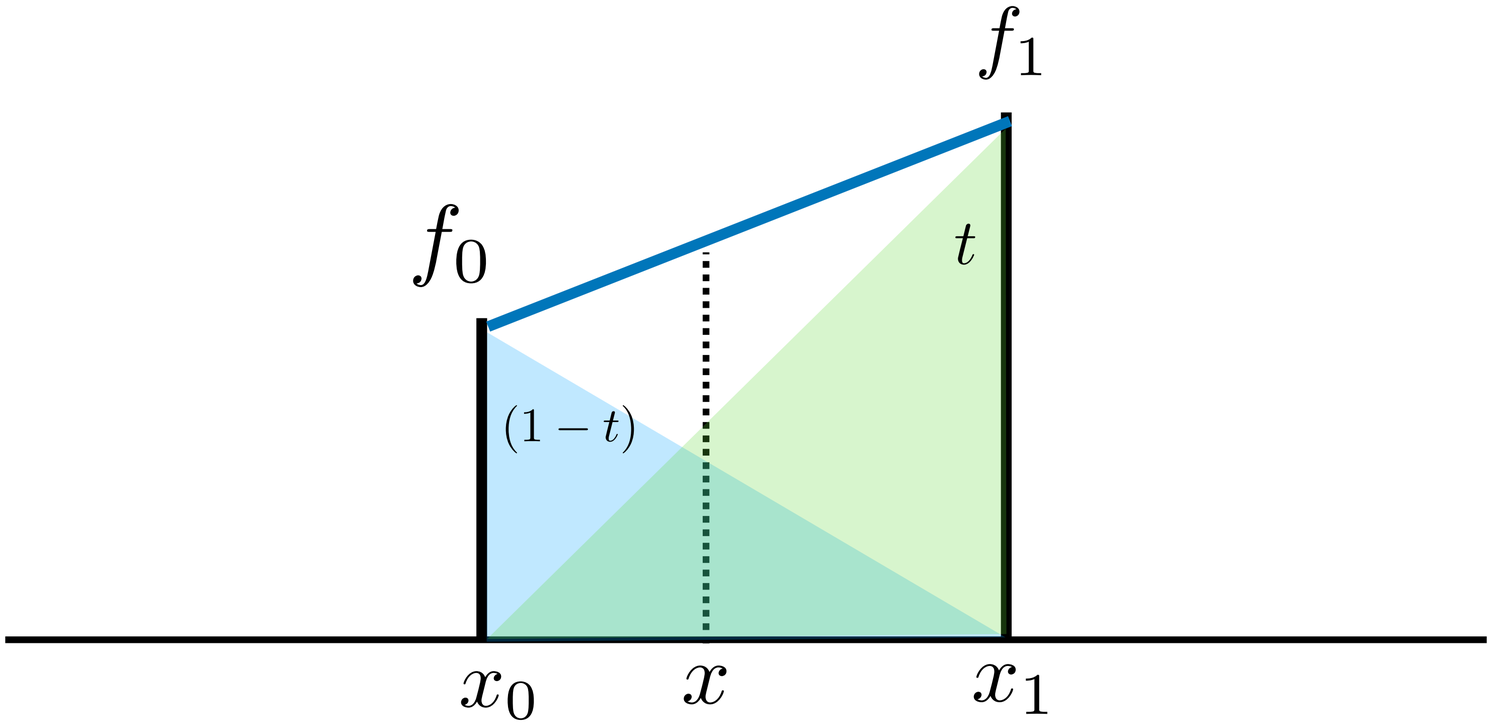


$$f(t) = (1-t)f_0 + tf_1$$

$$t = \frac{x - x_0}{x_1 - x_0}$$

Can think of linear interpolation as linear combination of two functions

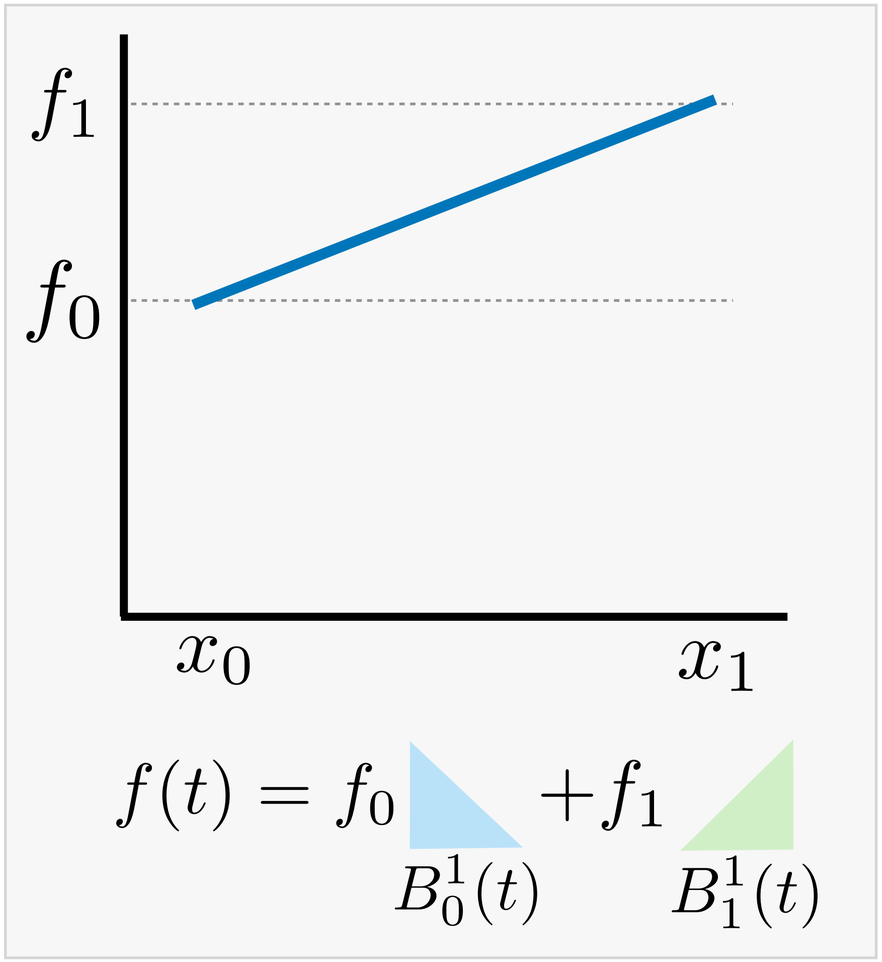
Weights are given by the two values (f_0 and f_1) being interpolated

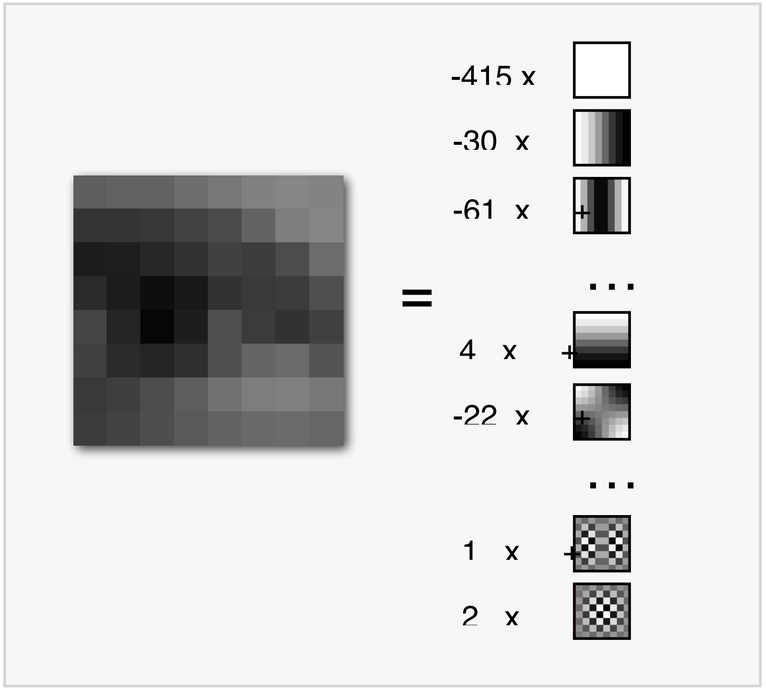


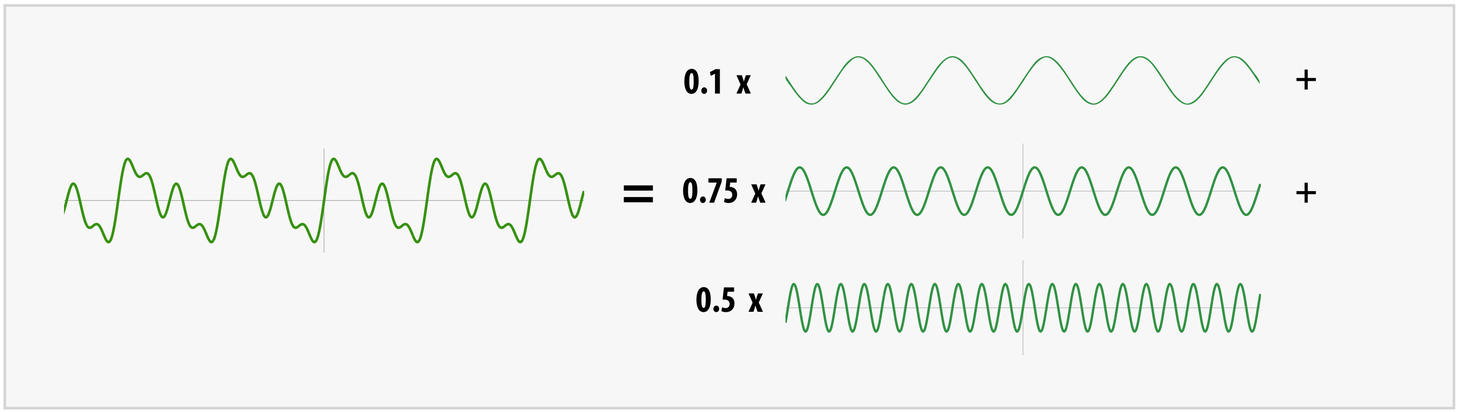
$$f(t) = (1-t)f_0 + tf_1$$

Note: this is the idea of representing a function in a new basis, again!

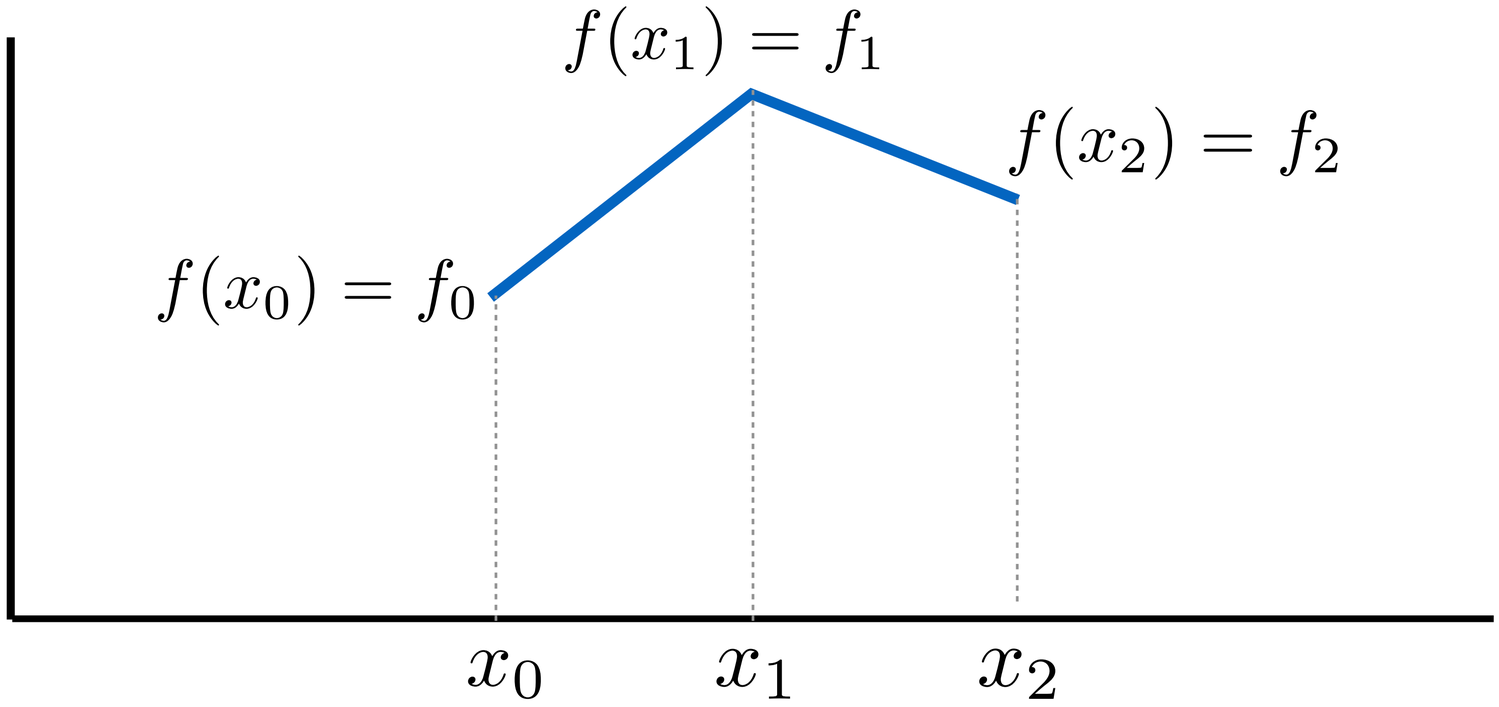
My function *f* is represented as a superposition (weighted sum) of a set of basis functions



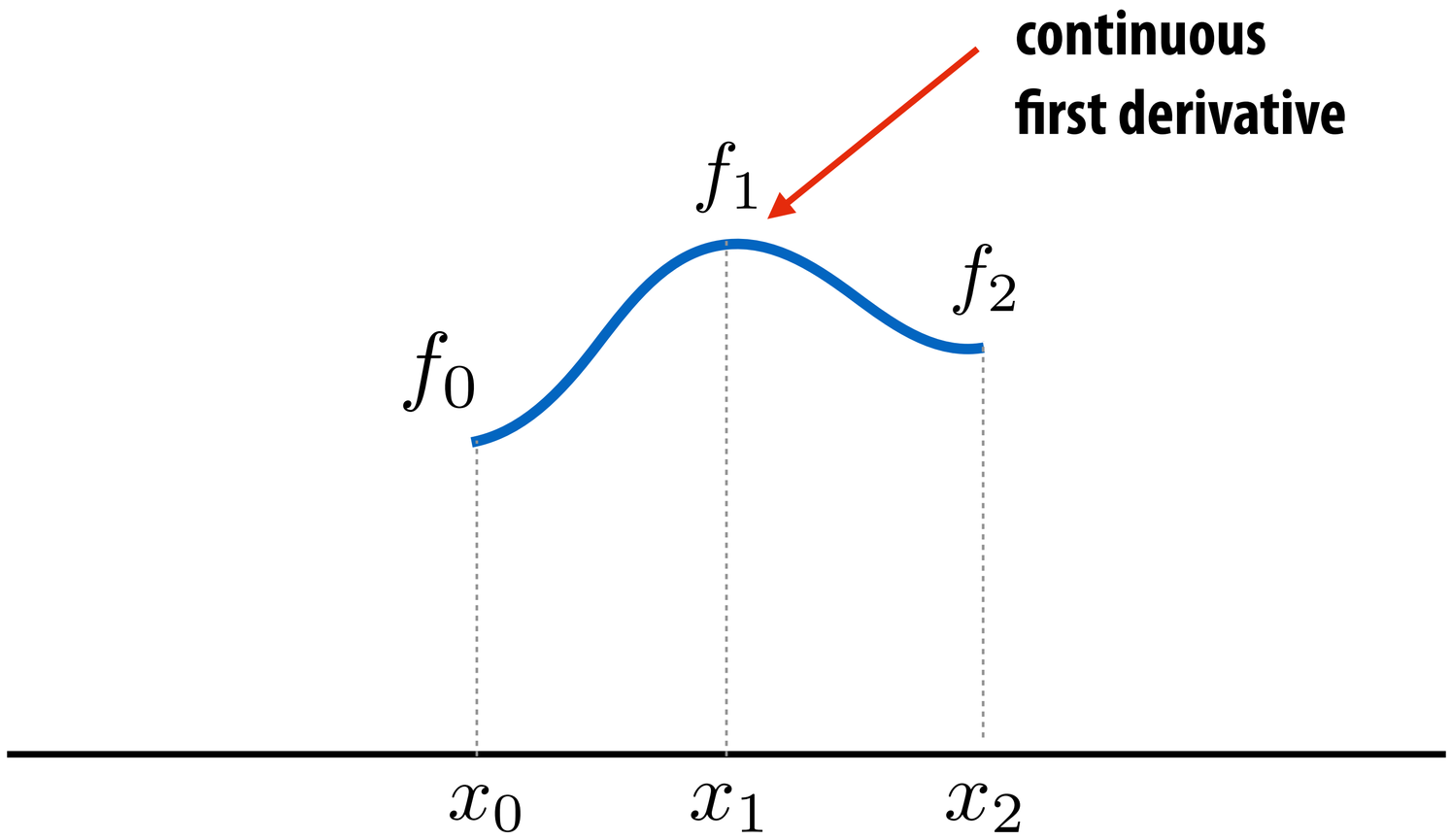




Problem with piecewise linear interpolation: derivatives are not continuous

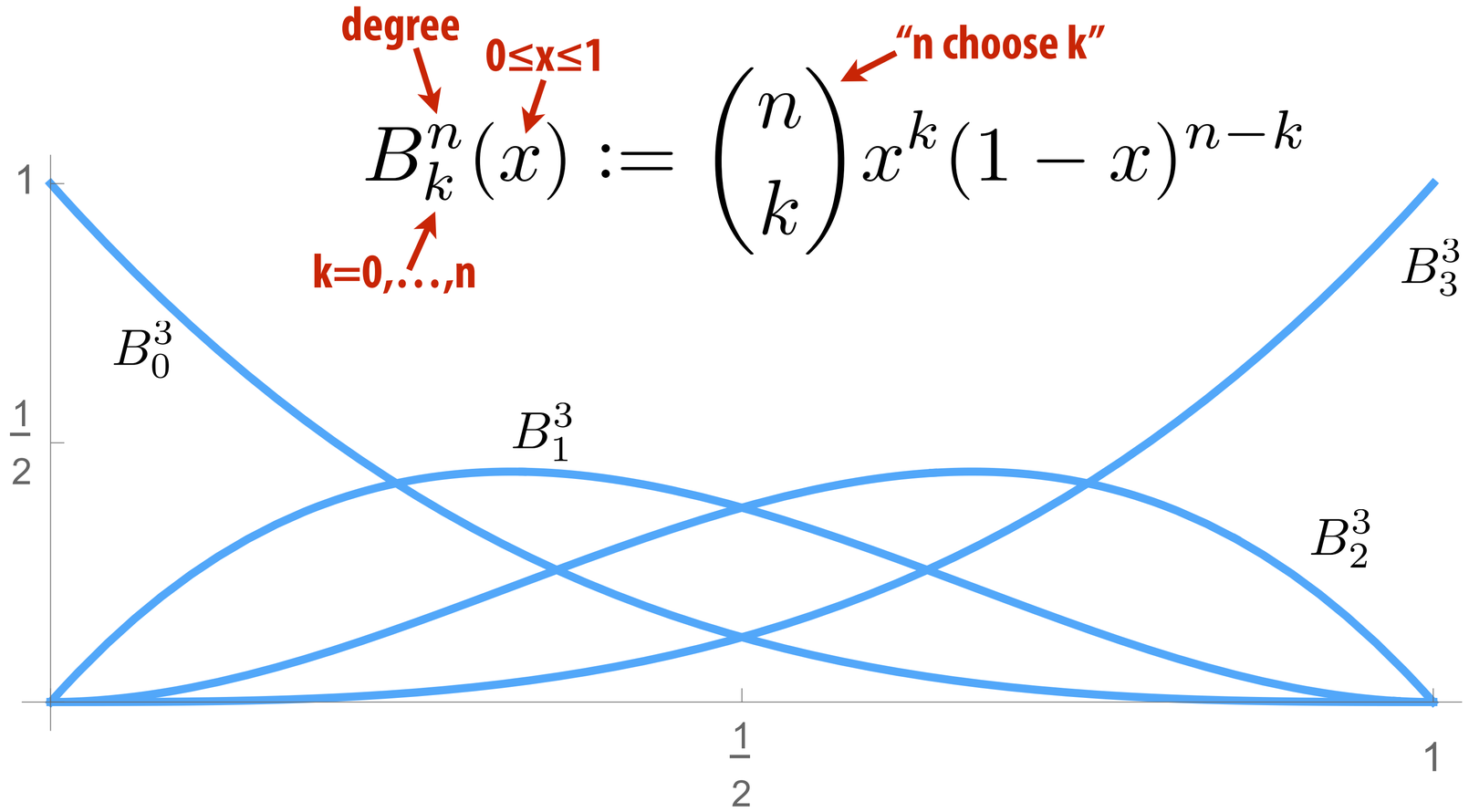


Smooth interpolation?



Bernstein basis

- Why limit ourselves to just linear interpolation?
- More flexibility by using higher-order polynomials
- Instead of usual basis (1, x, x^2, x^3, ...), use Bernstein basis:



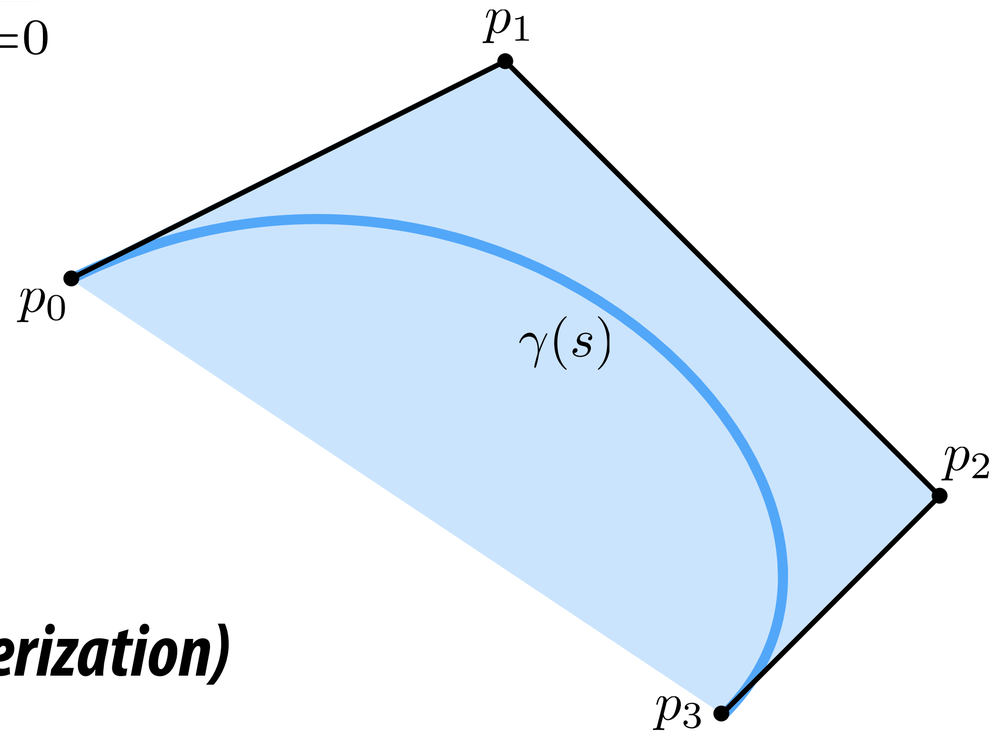
Bézier curves (explicit)

A Bézier curve is a curve expressed in the Bernstein basis:

$$\gamma(s) := \sum_{k=0}^{n} B_{n,k}(s) p_k$$

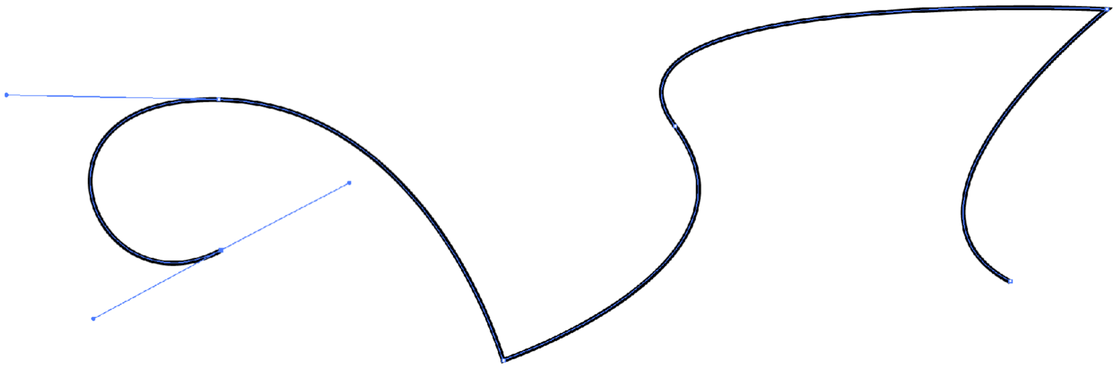
control points

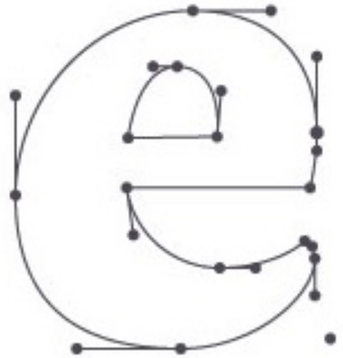
- For n=1, just get a line segment!
- For n=3, get "cubic Bézier":
- Important features:
  1. interpolates endpoints
  2. tangent to end segments
  3. contained in convex hull *(nice for rasterization)*



Piecewise Bézier curves (explicit)

- More interesting shapes: piece together many Bézier curves
- Widely-used technique (Illustrator, fonts, SVG, etc.)





- Formally, piecewise Bézier curve:

piecewise Bézier

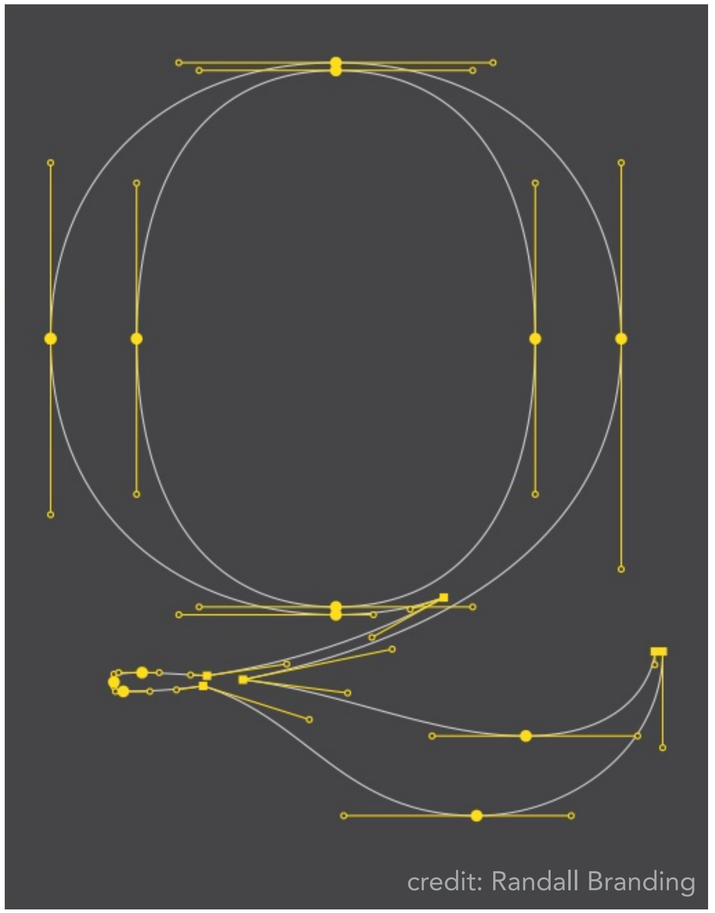
$$\gamma(u) := \gamma_i \left(\frac{u - u_i}{u_{i+1} - u_i} \right), \qquad u_i \leq u < u_{i+1}$$

single Bézier

Vector fonts

The Quick Brown
Fox Jumps Over
The Lazy Dog

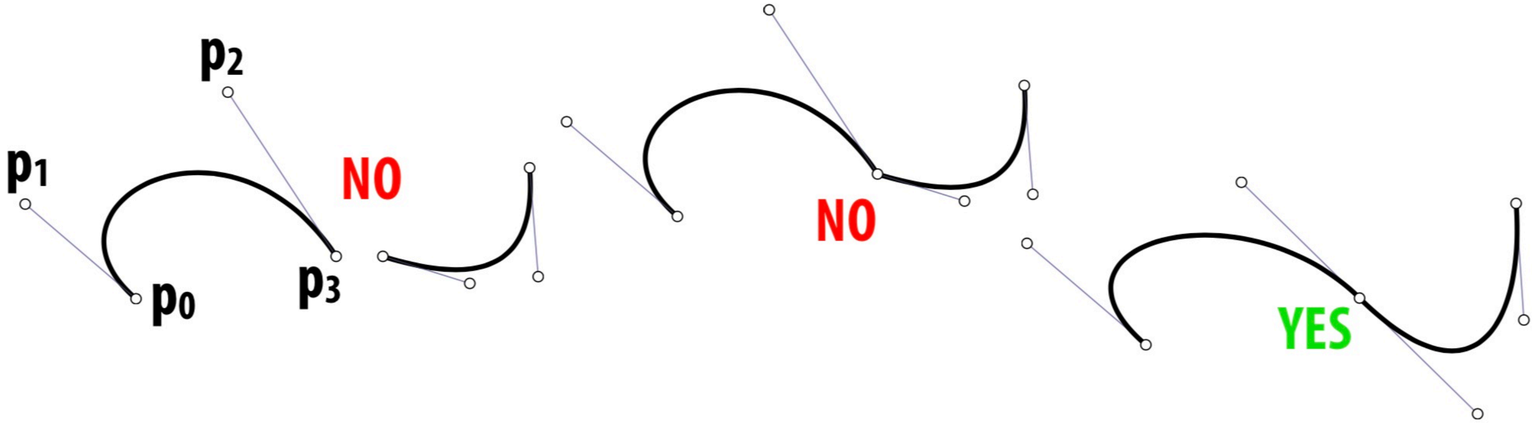
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 0123456789



Baskerville font - represented as cubic Bézier splines

Bézier curves — tangent continuity

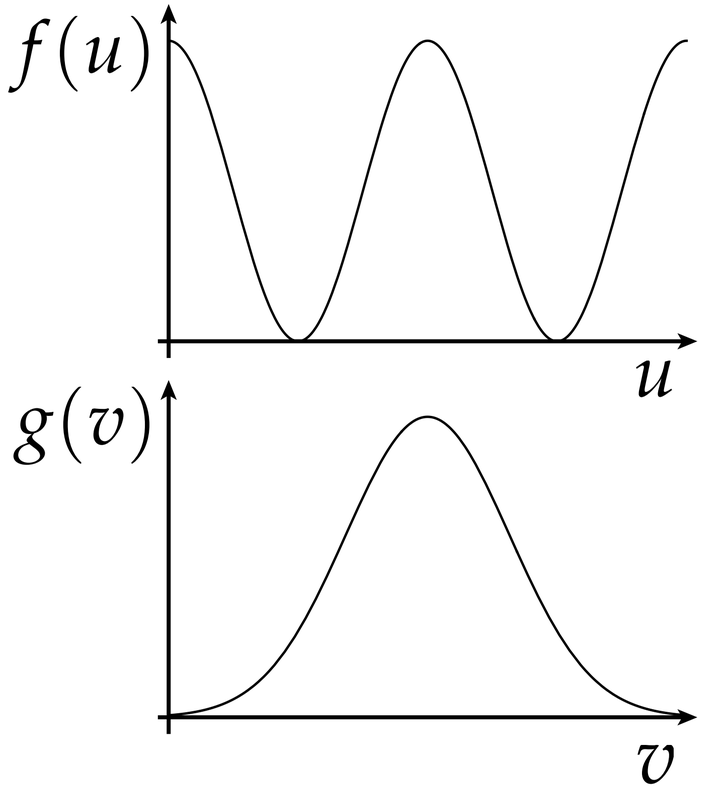
- To get "seamless" curves, want *points* and *tangents* to line up:

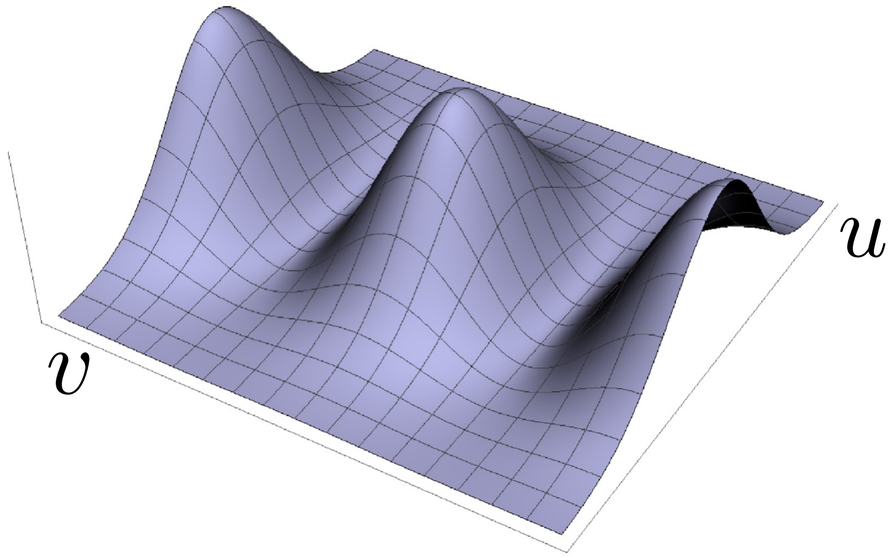


  - Ok, but how?
  - Each curve is cubic: $au^3 + bu^2 + cu + d$
  - Q: How many constraints vs. degrees of freedom?
  - Q: Could you do this with *quadratic* Bézier? *Linear* Bézier?

Tensor product

- Can use a pair of curves to get a surface
- Value at any point (u,v) given by product of a curve f(u) and a curve g(v) (sometimes called the *"tensor product"*):

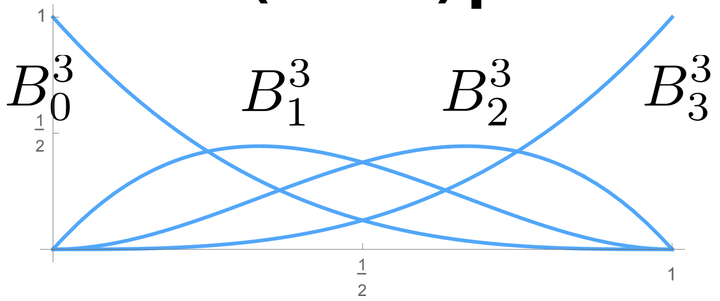




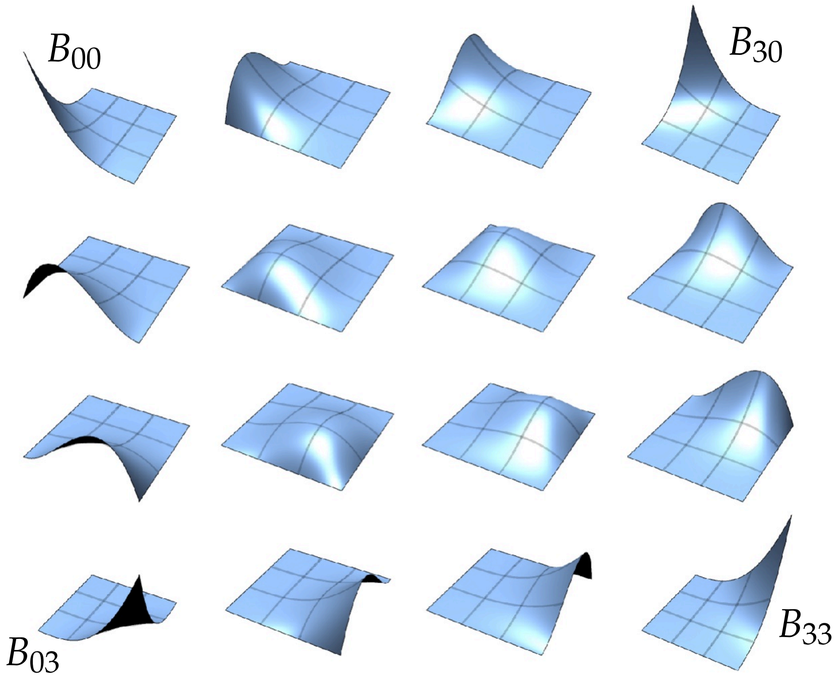
$$(f \otimes g)(u,v) := f(u)g(v)$$

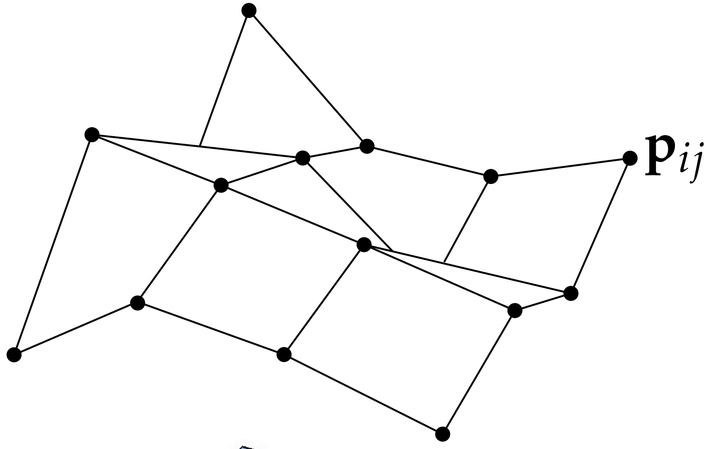
Bézier patches

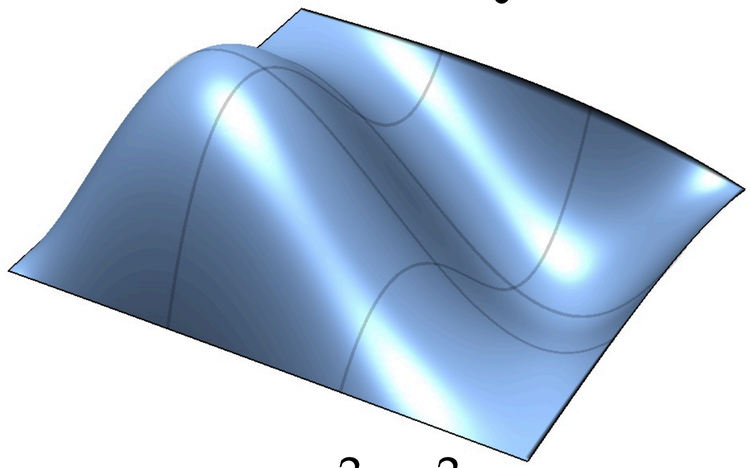
- ***Bézier patch*** **is sum of (tensor) products of Bernstein bases**



$$B_{i,j}^3(u,v) := B_i^3(u)B_j^3(v)$$



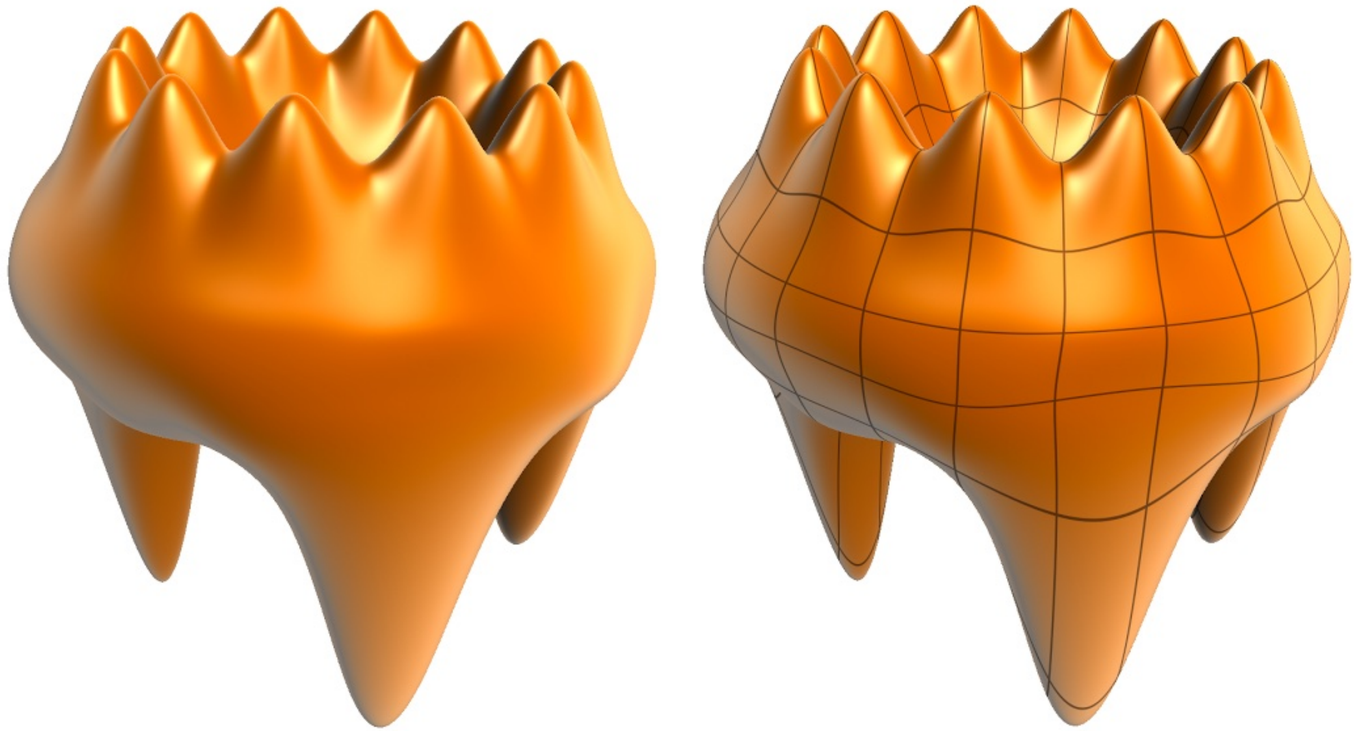




$$S(u,v) := \sum_{i=0}^{3}\sum_{j=0}^{3} B_{i,j}^3(u,v)\mathbf{p}_{ij}$$

Bézier surface

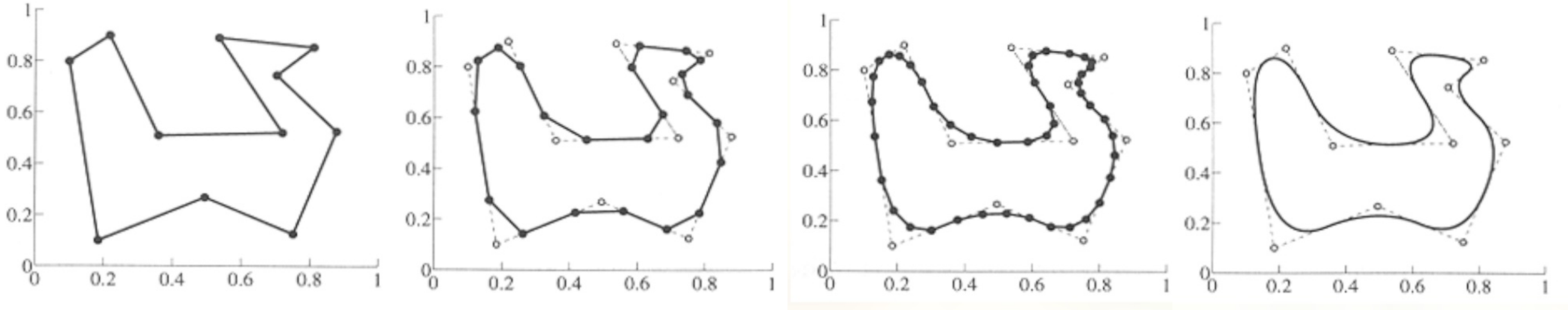
- Just as we connected Bézier *curves*, can connect Bézier *patches* to get a surface:



- *Very* easy to draw: just dice each patch into regular (u,v) grid!

Subdivision

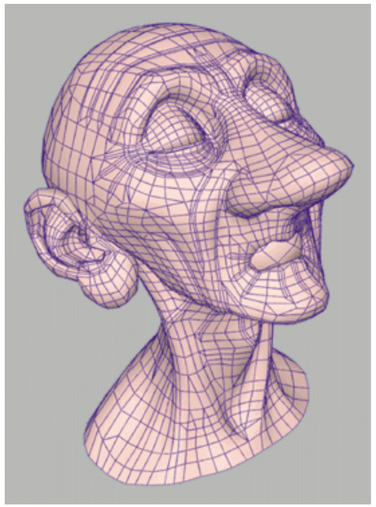
- Alternative starting point for curves/surfaces: *subdivision*
- Start with control curve
- Insert new vertex at each edge midpoint
- Update vertex positions according to fixed rule
- For careful choice of averaging rule, yields smooth curve
  - *Some subdivision schemes correspond to well-known spline schemes!*

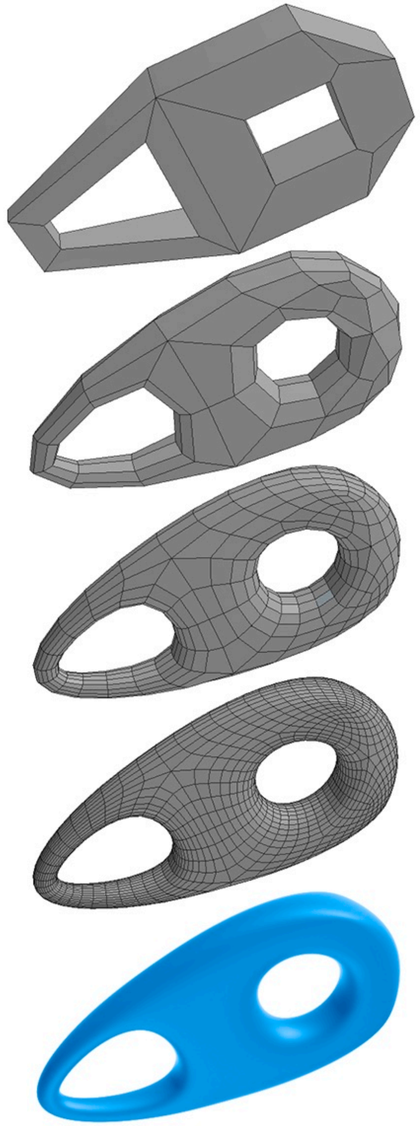


Slide cribbed from Don Fussell.

Subdivision surfaces (explicit)

- Start with coarse polygon mesh (“control cage”)
- Subdivide each element
- Update vertices via local averaging
- Many possible rules:
  - Catmull-Clark (for quad meshes)
  - Loop (for triangle meshes)
  - …
- Common issues:
  - interpolating or approximating?
  - continuity at vertices?
- Easier than splines for modeling; harder to evaluate pointwise



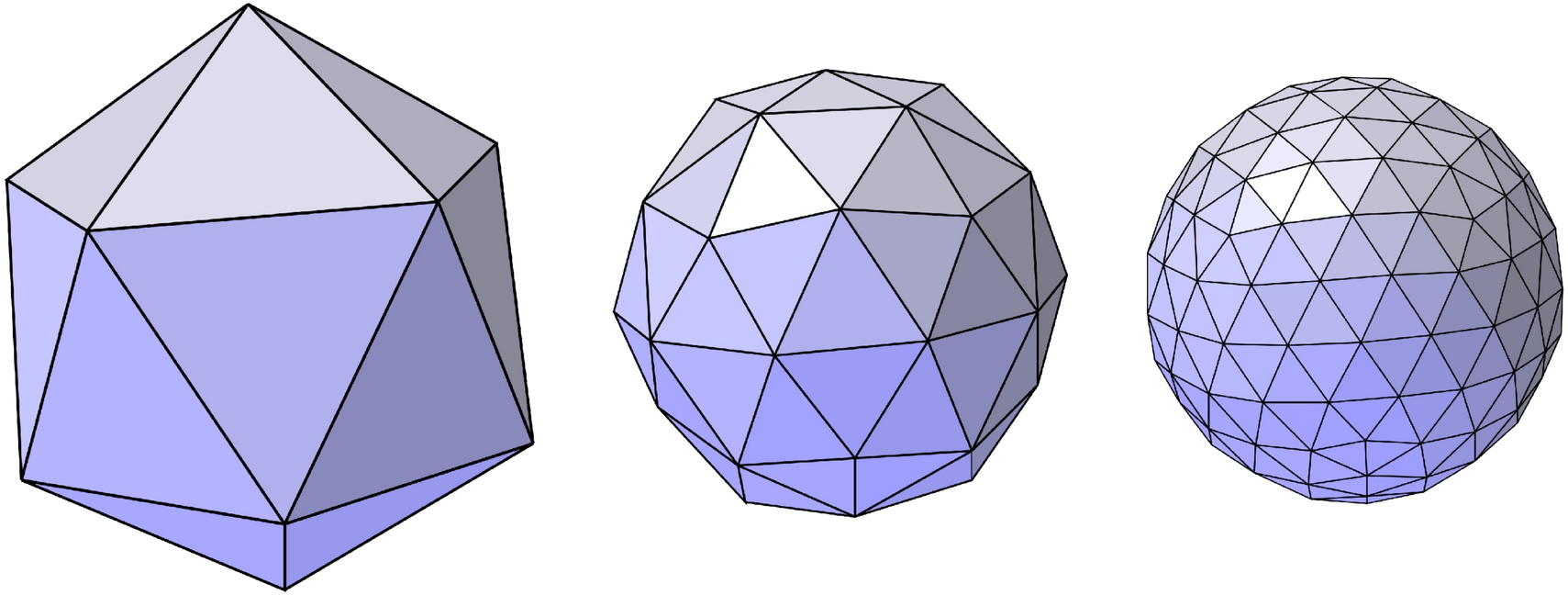


Loop subdivision results

Common subdivision rule for triangle meshes

"C2" smoothness away from "irregular" vertices, C1 everywhere else

Approximating, not interpolating



Credit: Simon Fuhrman

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

Acknowledgements

- Thanks to Keenan Crane and Ren Ng for slide materials