

Shape Representations

D.A. Forsyth, UIUC

Key ideas

- There are many representations of 3D shapes
 - each is good at some things, bad at others
 - one can usually move between representations
 - can be hard for some pairs
- Networks can make any of them
 - conventional paper:
 - I made a network whose input is
 - pic, multiple pics, range map
 - and whose output is
 - some shape representation that hasn't been tried yet

Point clouds

- Easy to:
 - measure
 - render (easyish)
- Hard to:
 - compute most geometric properties
 - (eg volume; recover curvature; find features)
 - occlusion
 - use near neighbor, etc
- Dubious properties:
 - usually either
 - massively redundant
 - or tricky to work with

Voxels and octrees

- Voxels:
 - break space into an even grid; place something in each box
 - usually, an indicator function (0-1), but can get more interesting
- Octree:
 - slightly more efficient structure
 - start with unit box;
 - subdivide into 8 children (halving each dimension)
 - for each child either
 - recur
 - put something in child and stop
 - savings may not be as big as you think
 - function could be represented in a variety of ways
 - values in leaves; wavelet-like representation

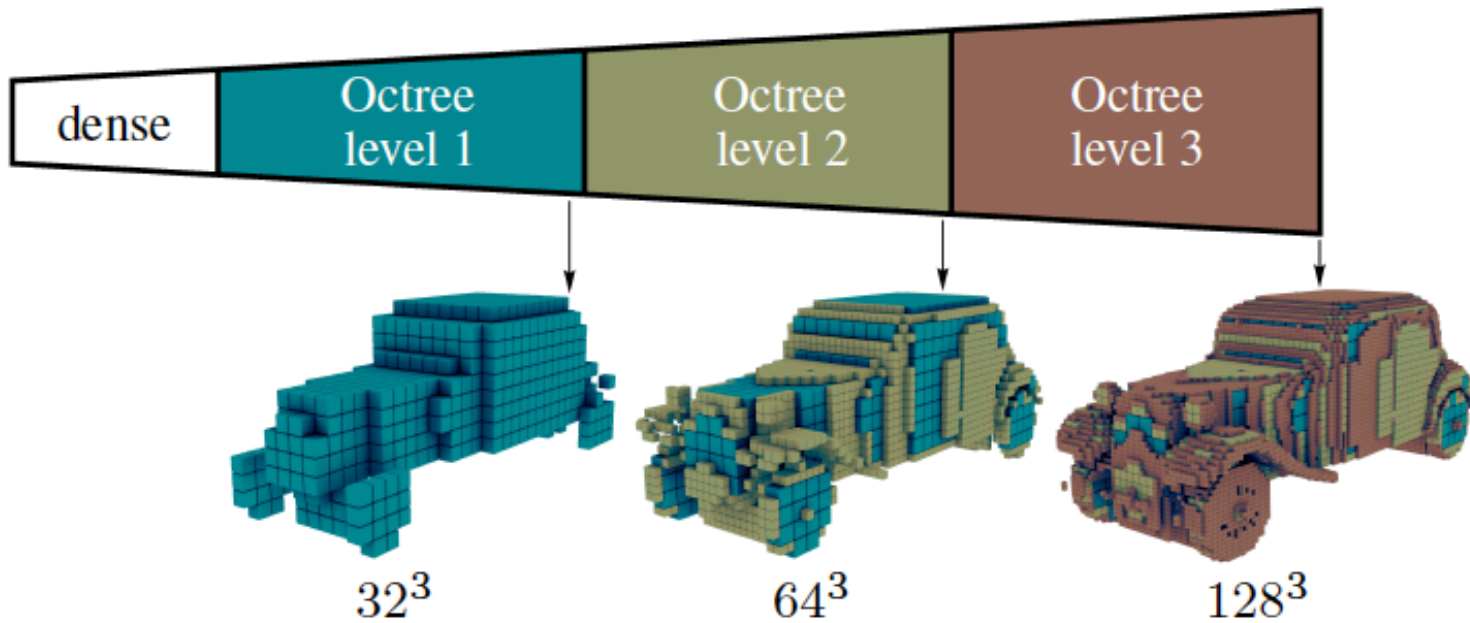
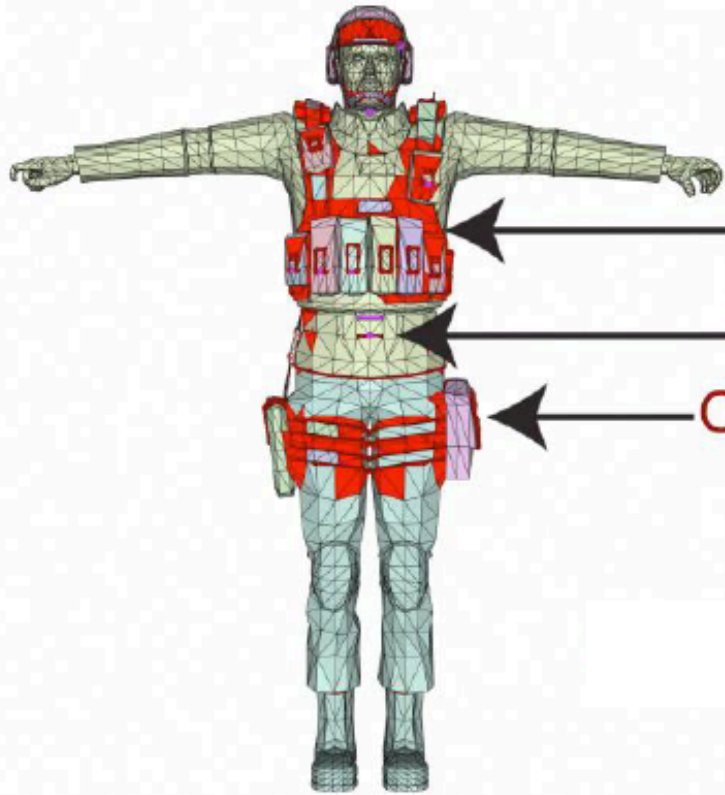


Figure 1. The proposed OGN represents its volumetric output as an octree. Initially estimated rough low-resolution structure is gradually refined to a desired high resolution. At each level only a sparse set of spatial locations is predicted. This representation is significantly more efficient than a dense voxel grid and allows generating volumes as large as 512^3 voxels on a modern GPU in a single forward pass.

Polygon soups and meshes

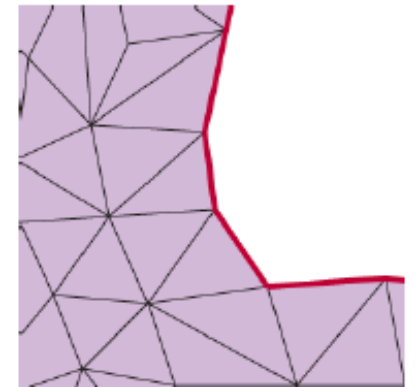
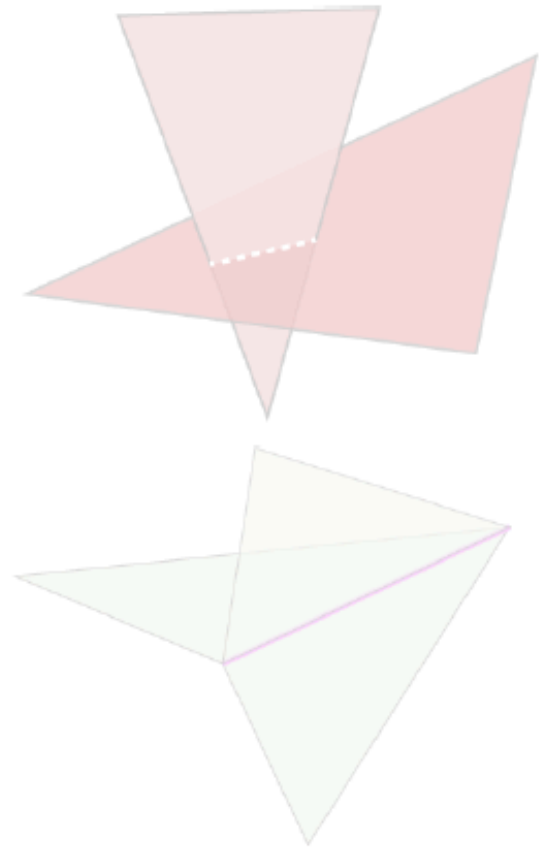
- Collection of polygons (usually triangles)
- Meshes:
 - polygons share some edges and vertices
 - often, not always, rules about how
 - eg in manifold meshes (pl manifolds) disallow some configurations
 - some rules make mesh rendering/representation much more efficient (eg triangle strips)
 - two structures:
 - combinatorial
 - which triangle has which vertex, which edge
 - geometric (embedding)
 - where the vertices are in 3D



Self-intersections

Nonmanifold edges

Open boundaries



Coros slides

Meshes

- Standard problems with known solutions:
 - constructing meshes from: point clouds; implicit surfaces; csg, etc.
 - simplifying meshes
- Easy to:
 - render (very fast)
 - compute normals, local geometric properties
 - smooth
- Nasty features:
 - not necessarily solid
 - tricky to tell if point is inside or out if it is
 - surface detail can require fine polygons and still be poorly represented
 - can be very large

Implicit surfaces

- Most general form:

$$f(x, y, z; \theta) = 0$$

- Important cases:
 - algebraic surfaces
 - f polynomial
 - composite surfaces
 - eg metaballs; f is a weighted sum of shifted primitives


Implicit surfaces

- Standard problems with known solutions
 - meshing: pass from implicit surface to mesh
 - not straightforward, I'll sketch issues
 - rendering: ray trace OR mesh
- Easy (ish)
 - meshing; rendering
 - forming solids
- Nasty features
 - hard to know number of connected components
 - can be tricky to fit to data
 - easiest: data is well sampled point cloud, fit classifier
 - depends on parametrization

Marching cubes (sketch)

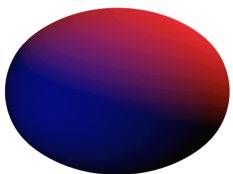
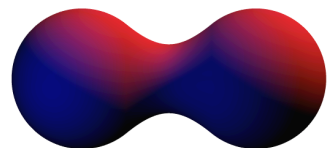
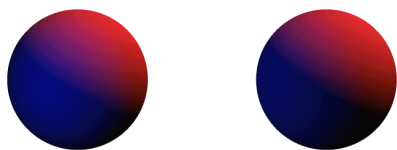
- Key ideas:
 - for small enough box, replace f with trilinear interpolate across box
 - there is mischief in this assumption
 - the mesh inside the box now takes a small set of possible patterns
 - indexed by the sign on the vertices; at most 256
 - symmetry etc. reduces the number of patterns
 - current estimate is 33
- Fast, efficient, practical, largely right

Composite surfaces

$$\sum_{i=1}^m c_i f(x - x_i, y - y_i, z - z_i; \theta_i) = t$$


- Metaballs
 - Choice of f matters; want
 - cheap to evaluate
 - local support
 - smooth
 - Common choices
 - $1/(r^2+e)$; $1/(r^2)^2$; gaussian
 - Original metaballs did not have c_i

1



2

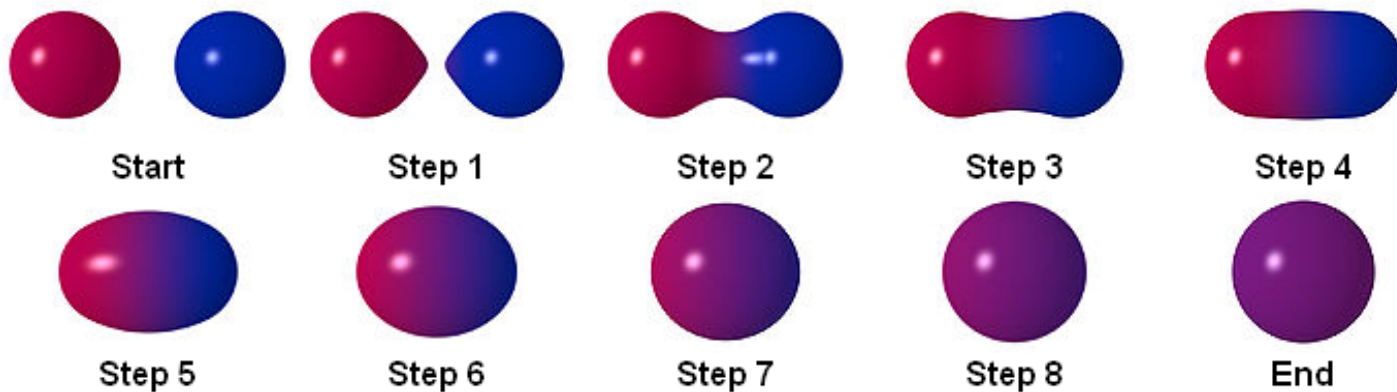
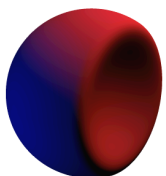
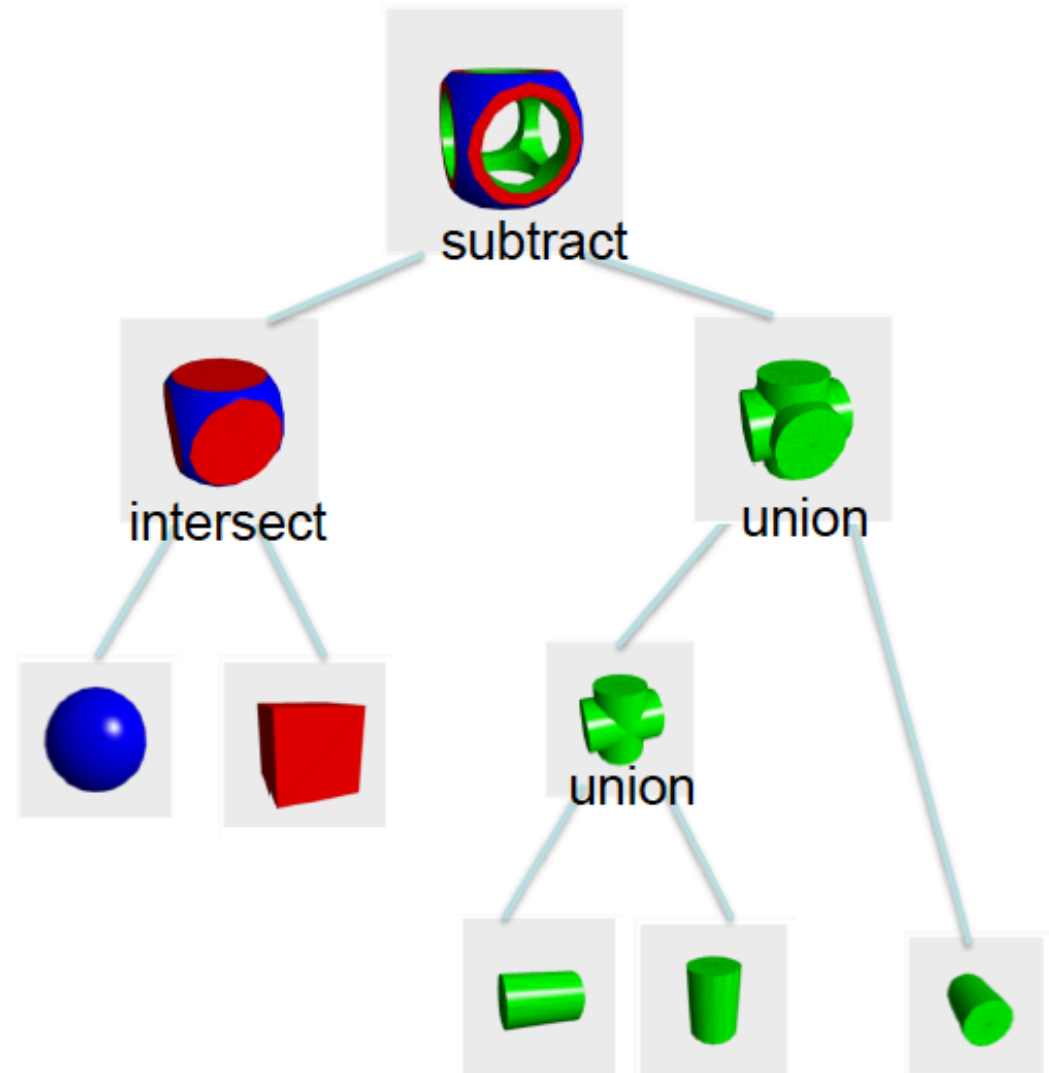


Figure credit: Wikipedia

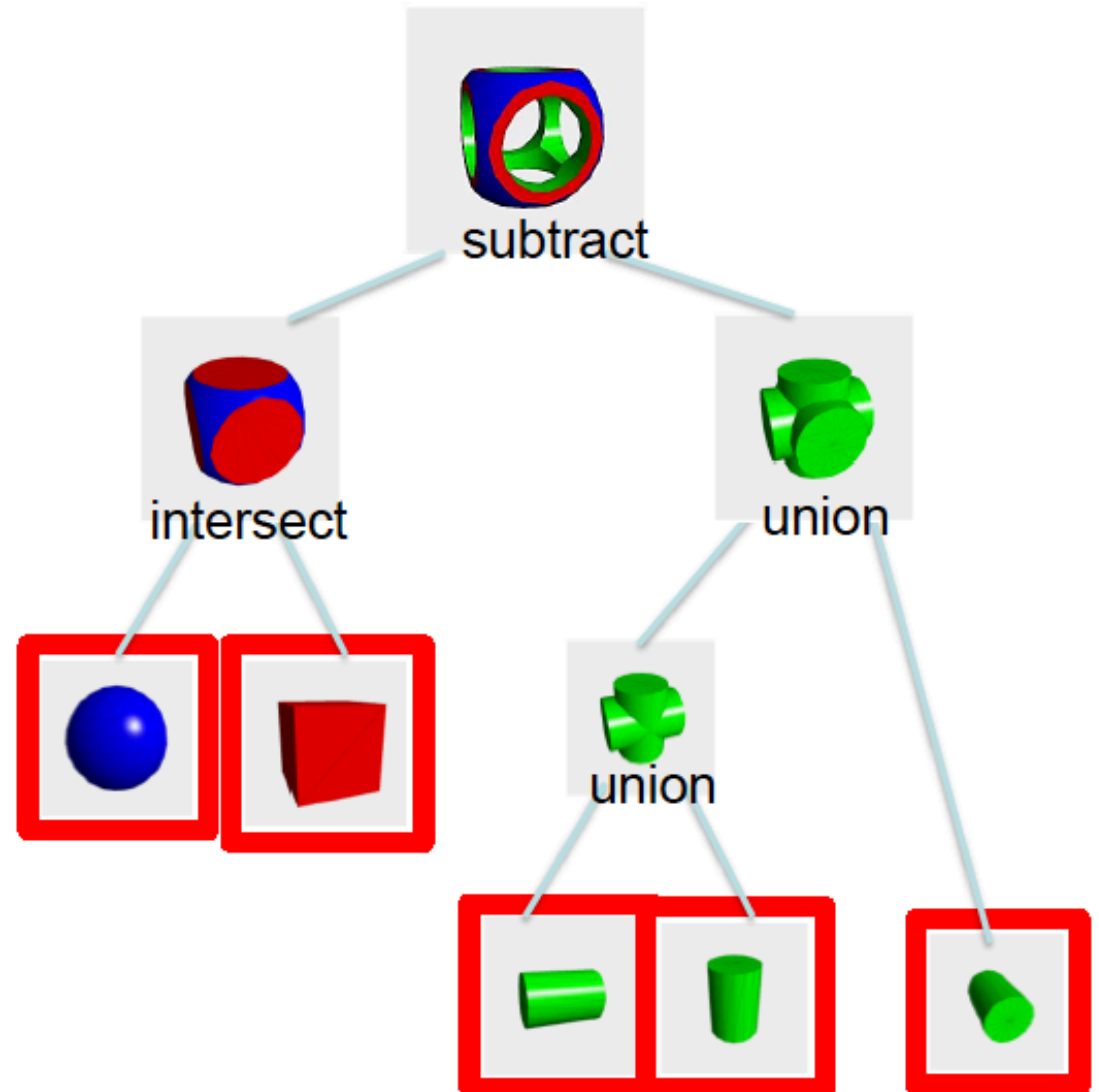
CSG Data Structure

- Stored in a Binary Tree data structure



Leaves: CSG Primitives

- Simple shapes
 - Cuboids
 - Cylinders
 - Prisms
 - Pyramids
 - Spheres
 - Cones
- Extrusions
- Surfaces of revolution
- Swept surfaces



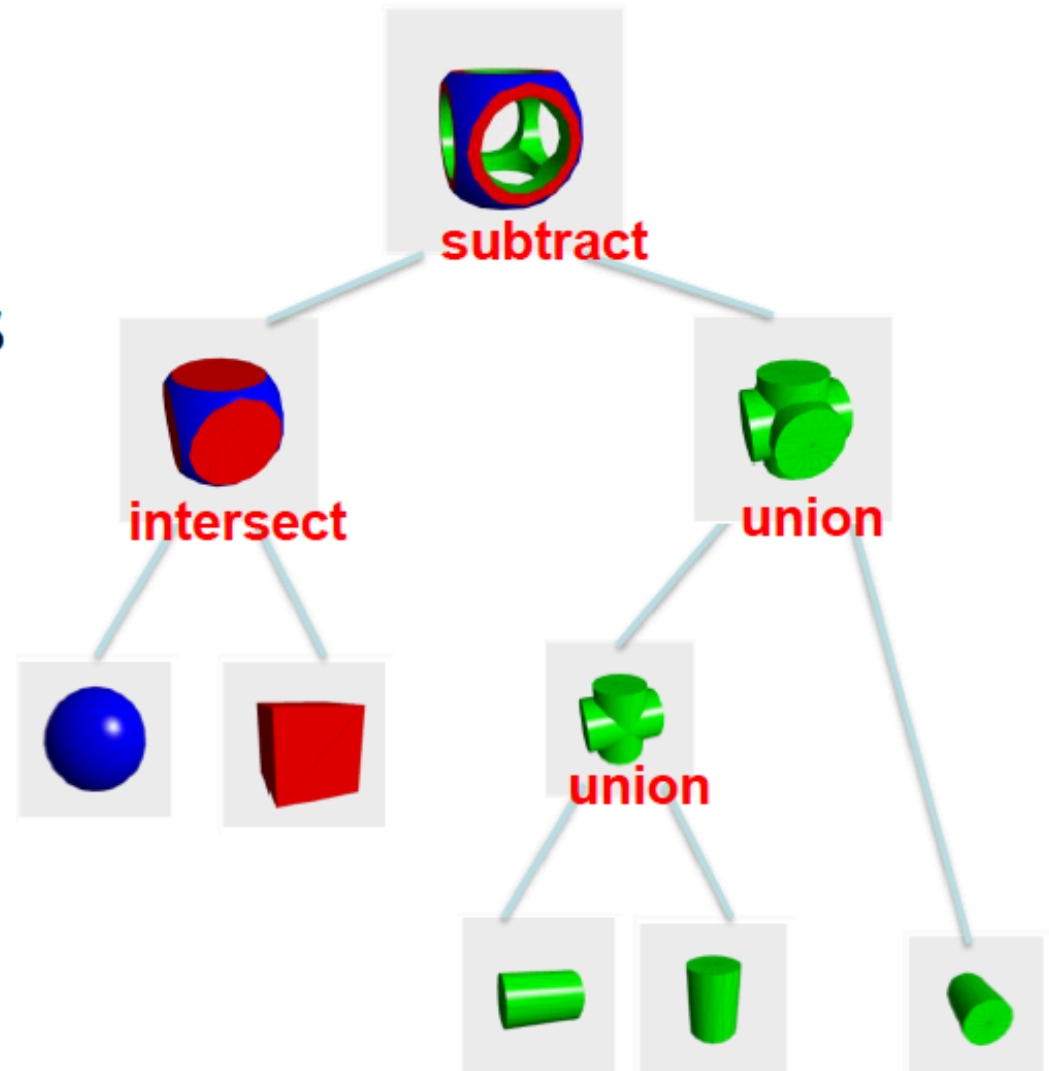
Internal Nodes

- Boolean Operations

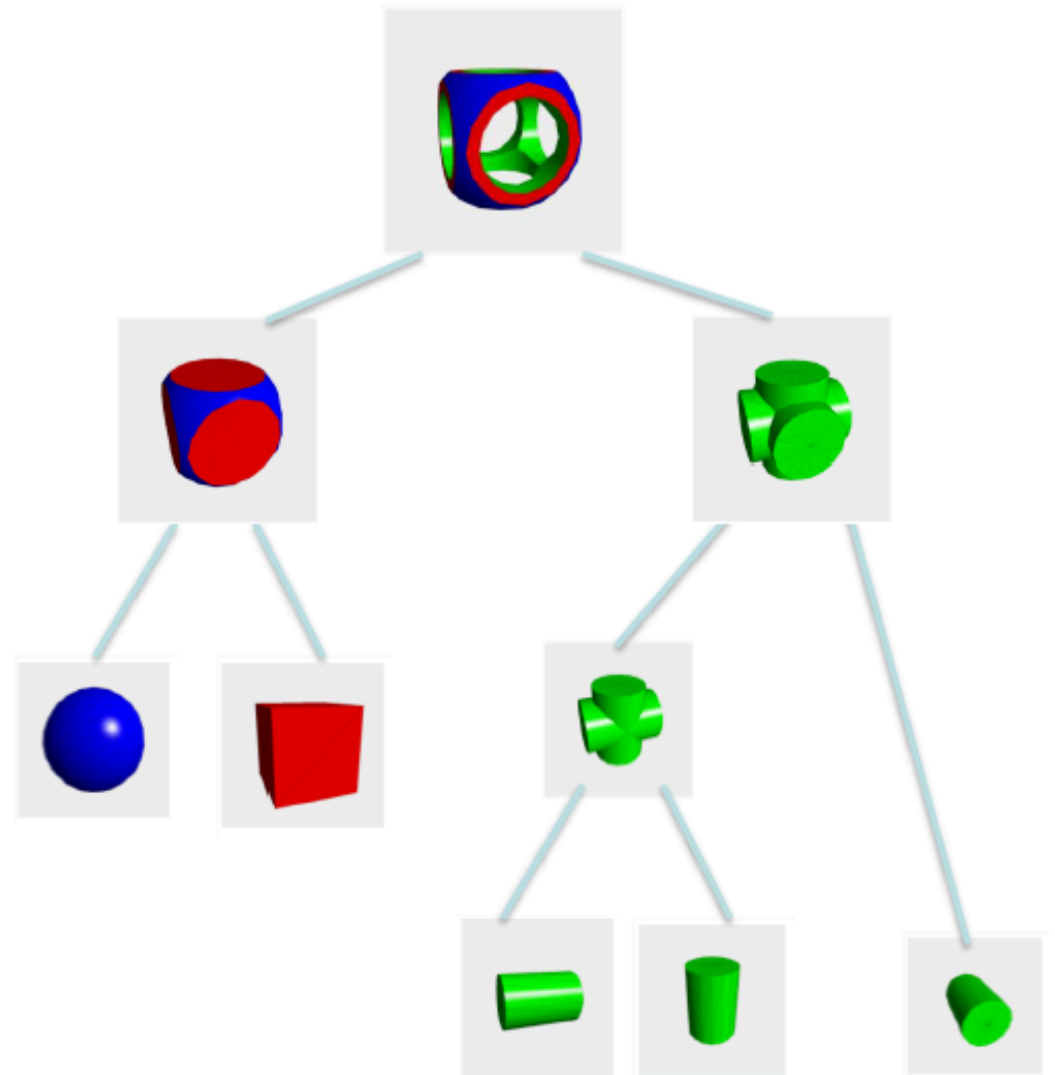
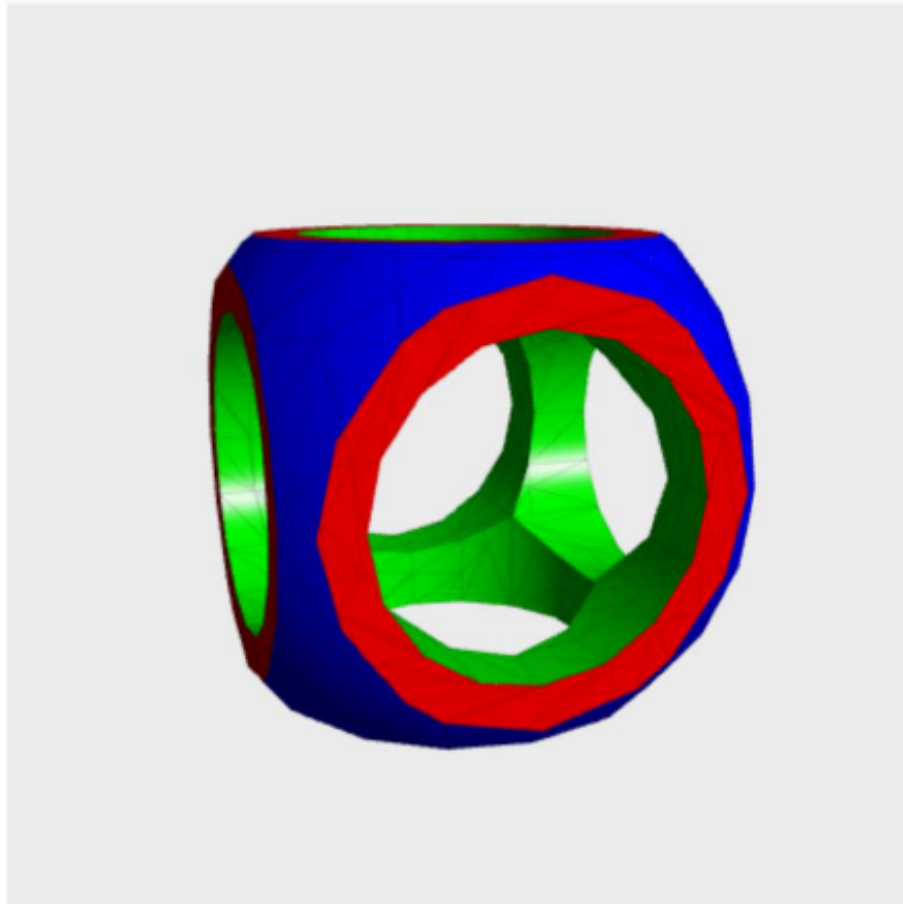
- Union
- Intersection
- Difference

- Rigid Transformations

- Scale
- Translation
- Rotation
- Shear

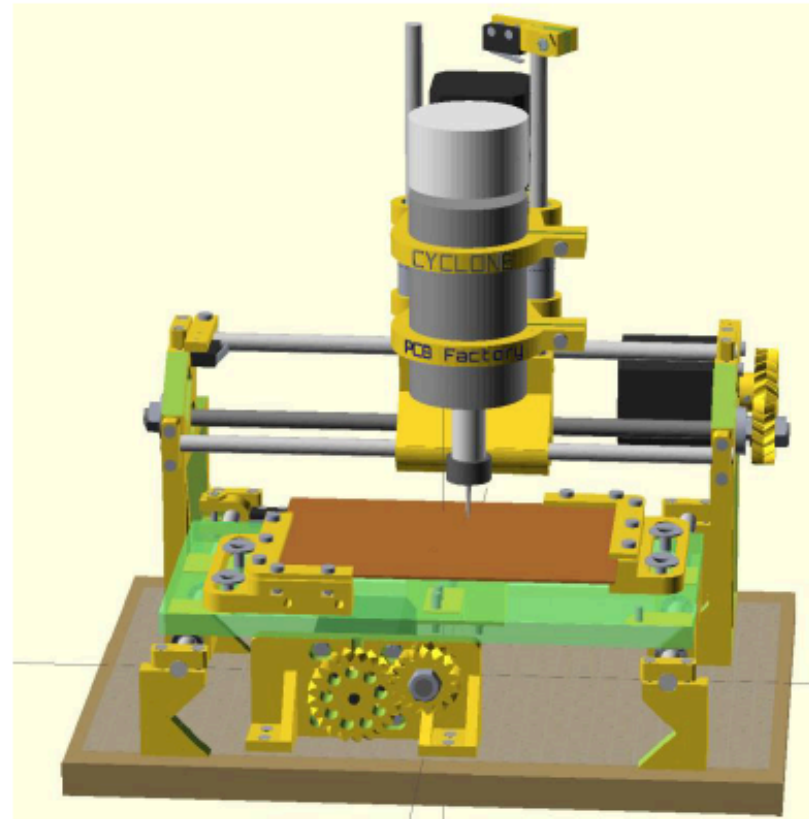


Root: The Final Object



OpenSCAD

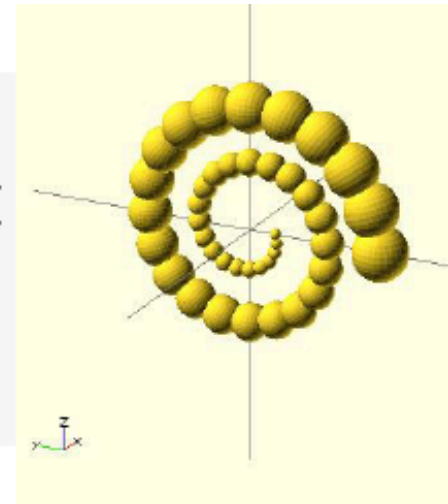
- Software for creating solid 3D CAD models
- Not an interactive modeler
 - Very basic UI
- A 3D-compiler
 - Geometry written as a script
 - Executed using CGAL/OpenCSG
 - Rendered with OpenGL
- Available for Linux/UNIX, Windows, Mac OS X
 - <http://www.openscad.org>



Procedural modelling

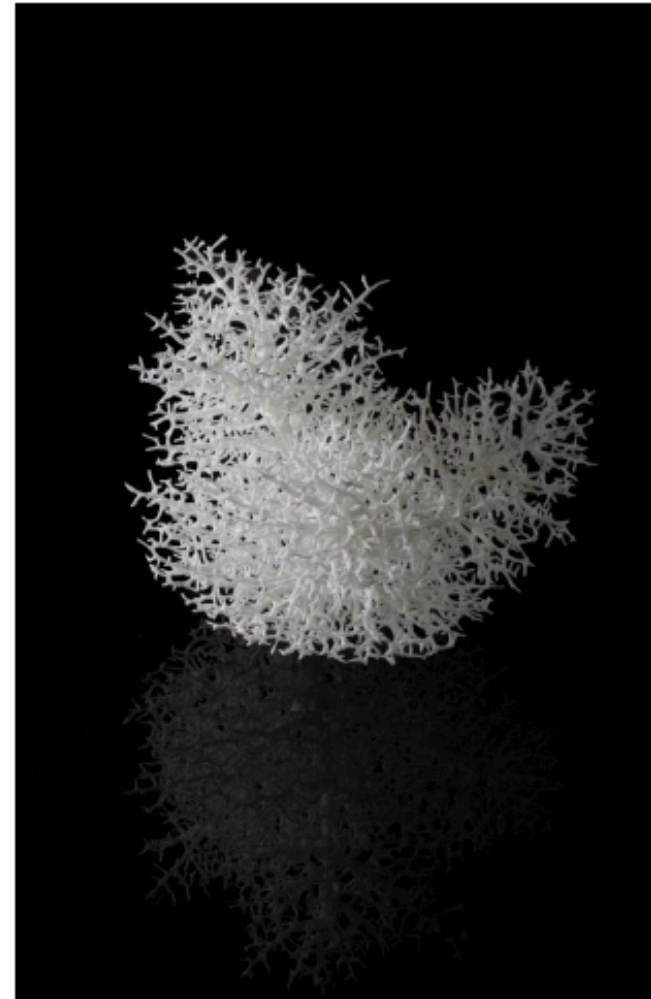
- Idea:
 - make CSG tree out of “program”

```
for (i = [10:50])  
  assign (angle = i*360/20, distance = i*10, r = i*2) {  
    rotate(angle, [1, 0, 0])  
    translate( [0, distance, 0] ) sphere(r = r);  
  }
```



Procedural Modeling

- Goal:
 - Describe 3D models algorithmically
- Best for models resulting from ...
 - Repeating or similar structures
 - Random processes
- Advantages:
 - Automatic generation
 - Concise representation
 - Parameterized classes of models



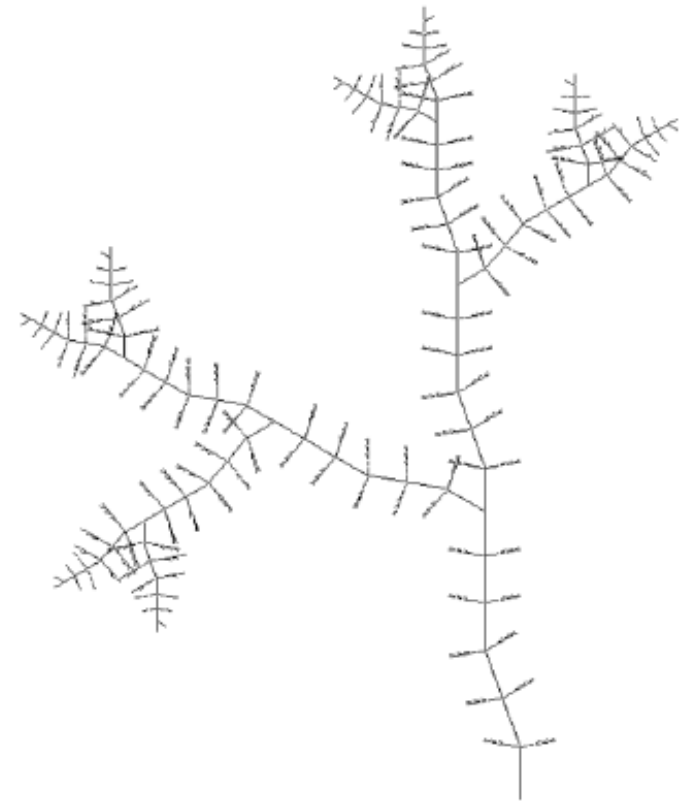
Formal Grammars and Languages

- A finite set of **nonterminal symbols**: $\{S, A, B\}$
- A finite set of **terminal symbols**: $\{a, b\}$
- A finite set of **production rules**: $S \rightarrow AB; A \rightarrow aBA$
- A **start symbol**: S

- Generates a set of finite-length sequences of symbols by recursively applying production rules starting with S

L-systems (Lindenmayer systems)

- A model of morphogenesis, based on formal grammars (set of rules and symbols)
- Introduced in 1968 by the Swedish biologist A. Lindenmayer
- Originally designed as a formal description of the development of simple multi-cellular organisms
- Later on, extended to describe higher plants and complex branching structures



L-system Example

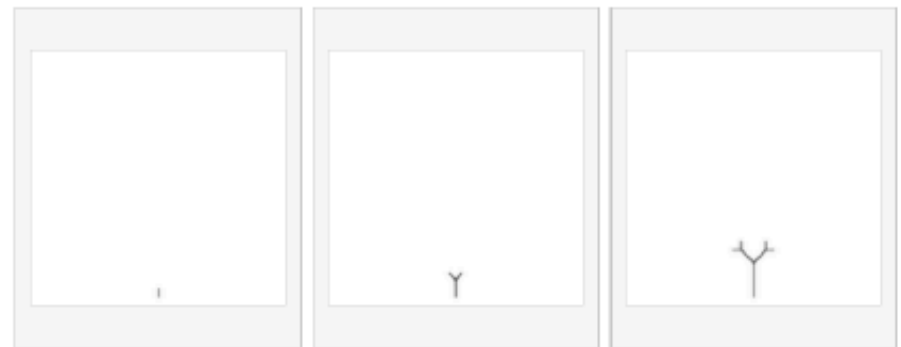
- **nonterminals** : 0, 1
- **terminals** : [,]
- **start** : 0
- **rules** : $(1 \rightarrow 11)$, $(0 \rightarrow 1[0]0)$

How does it work?

start: 0
1st recursion: 1[0]0
2nd recursion: 11[1[0]0]1[0]0
3rd recursion: 1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0

L-system Example

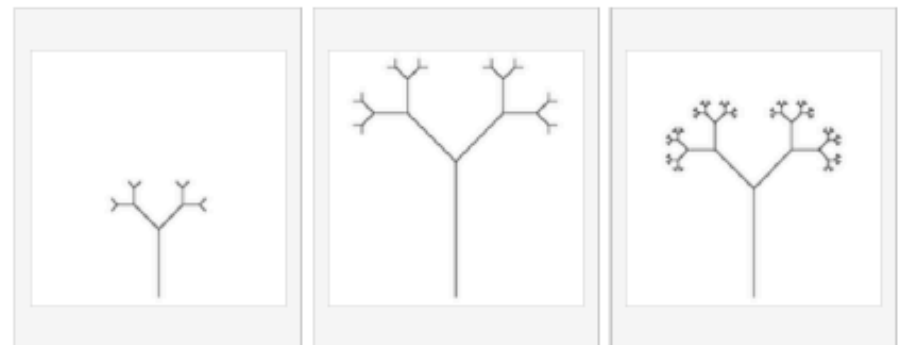
- **Visual representation: turtle graphics**
 - **0**: draw a line segment ending in a leaf
 - **1**: draw a line segment
 - **[**: push position and angle, turn left 45 degrees
 - **]**: pop position and angle, turn right 45 degrees



Axiom

First recursion

Second recursion



Third recursion

Fourth recursion

Seventh recursion, scaled down ten times

start: 0
1st recursion: 1[0]0
2nd recursion: 11[1[0]0]1[0]0
3rd recursion: 1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0

Applications: Plant Modeling

- Algorithmic Botany @ the University of Calgary
 - Covers many variants of L-Systems, formal derivations, and exhaustive coverage of different plant types.
 - <http://algorithmicbotany.org/papers>
 - http://algorithmicbotany.org/virtual_laboratory/

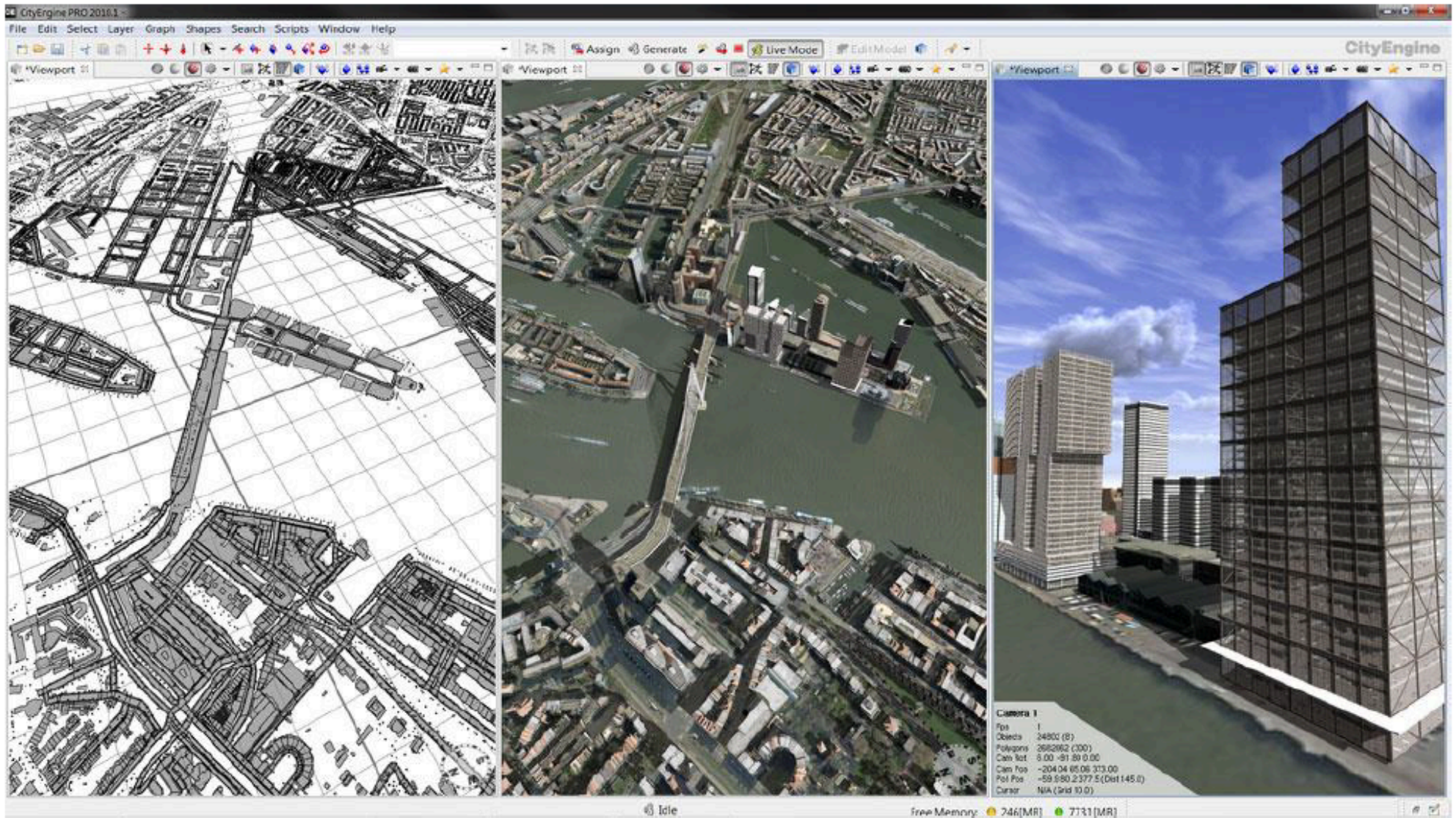


Procedural Modeling of Buildings

- Pompeii

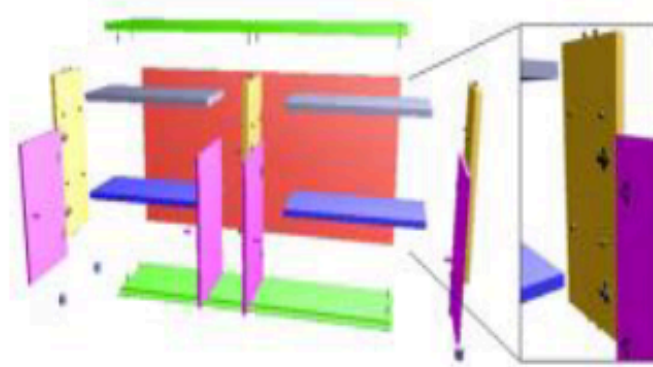
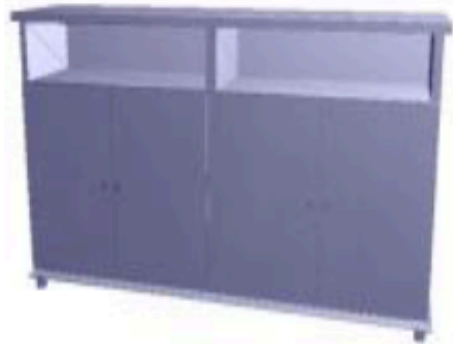


CityEngine

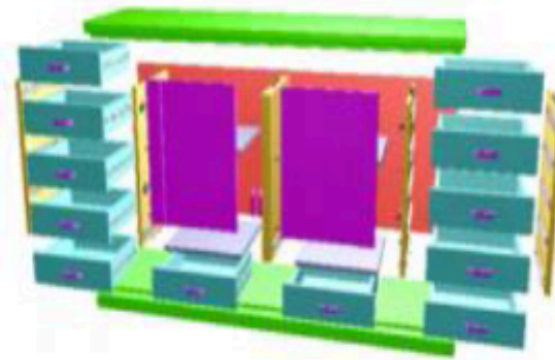


<http://www.esri.com/software/cityengine/>

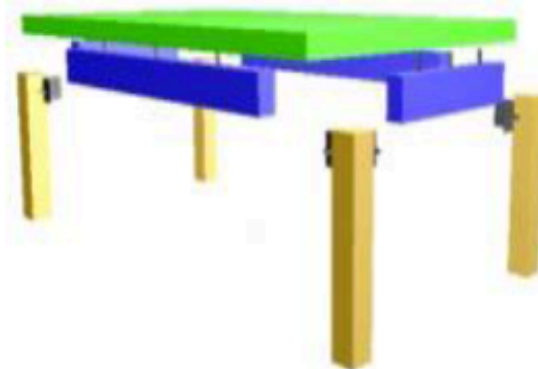
Furniture Design



Input:
3D
model



Output:
Fabricatable
Parts and
Connectors



Natural problems

- From input create model
 - inputs: point cloud, depth map, image, images, etc
 - model: in one of these forms
- From format A, create format B
 - mostly covered in classical literature
- From 3D rep'n, segment into semantic components
- From 3D rep'n, impute CSG
- From many 3D examples, impute procedural model

Pointnet - a neat trick

- Required: learned feature representation of a point cloud
- Difficulty: point cloud has no order
 - you can get the same point cloud in a different order
 - could impose order, but...
- Permutation invariants:
 - the basis for permutation invariants are the symmetric functions
 - mostly, a nuisance to work with
- Idea:
 - for any point cloud of n points in d dimensions,

$$\begin{bmatrix} \max(x_{1,1}, x_{2,1}, \dots, x_{n,1}) \\ \dots \\ \max(x_{1,d}, x_{2,d}, \dots, x_{n,d}) \end{bmatrix}$$

is permutation invariant

Pointnet - a neat trick - II

- So:
 - embed points in high dimension (K)
 - compute this pooling
 - now compute embedding of this feature vector
 - the resulting object is permutation invariant
 - and “general”
 - assume
 - $f(S)$ continuous in hausdorff distance on point sets
 - hausdorff distance on point sets = max dist to nearest neighbor
 - choose eps, and K big enough
 - then there is some $g(S)$ of this form st $|f(S)-g(S)| < \text{eps}$

Formally...

Theorem 1. *Suppose $f : \mathcal{X} \rightarrow \mathbb{R}$ is a continuous set function w.r.t Hausdorff distance $d_H(\cdot, \cdot)$. $\forall \epsilon > 0$, \exists a continuous function h and a symmetric function $g(x_1, \dots, x_n) = \gamma \circ \text{MAX}$, such that for any $S \in \mathcal{X}$,*

$$\left| f(S) - \gamma \left(\text{MAX}_{x_i \in S} \{h(x_i)\} \right) \right| < \epsilon$$

where x_1, \dots, x_n is the full list of elements in S ordered arbitrarily, γ is a continuous function, and MAX is a vector max operator that takes n vectors as input and returns a new vector of the element-wise maximum.

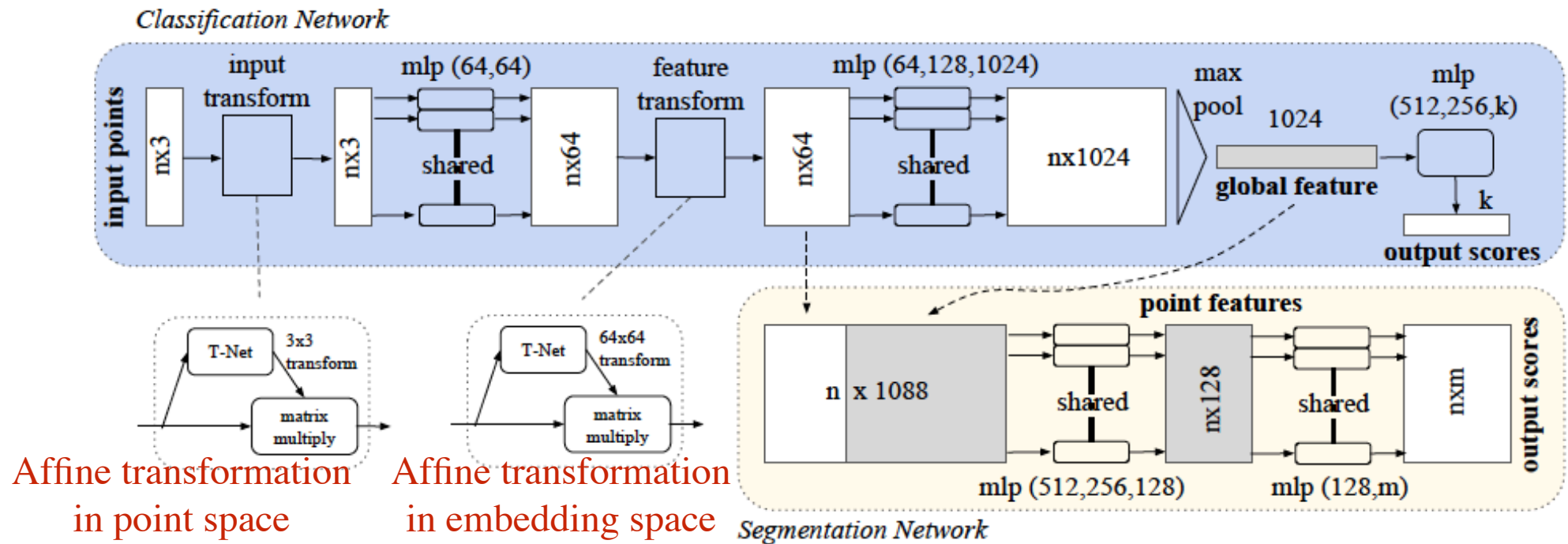


Figure 2. **PointNet Architecture.** The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. “mlp” stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

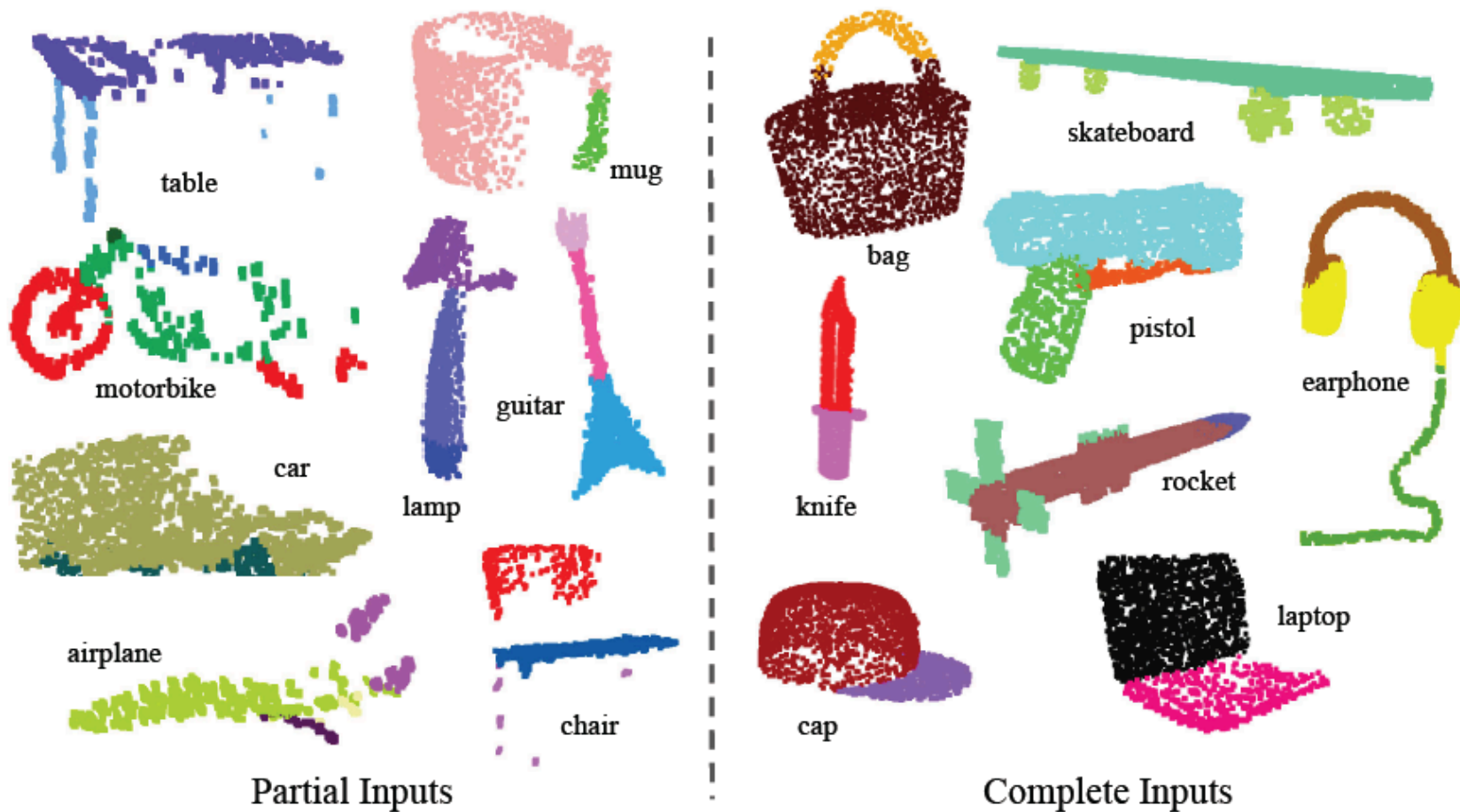


Figure 3. **Qualitative results for part segmentation.** We visualize the CAD part segmentation results across all 16 object categories. We show both results for partial simulated Kinect scans (left block) and complete ShapeNet CAD models (right block).



Figure 4. **Qualitative results for semantic segmentation.** Top row is input point cloud with color. Bottom row is output semantic segmentation result (on points) displayed in the same camera viewpoint as input.

Does this measure local properties?

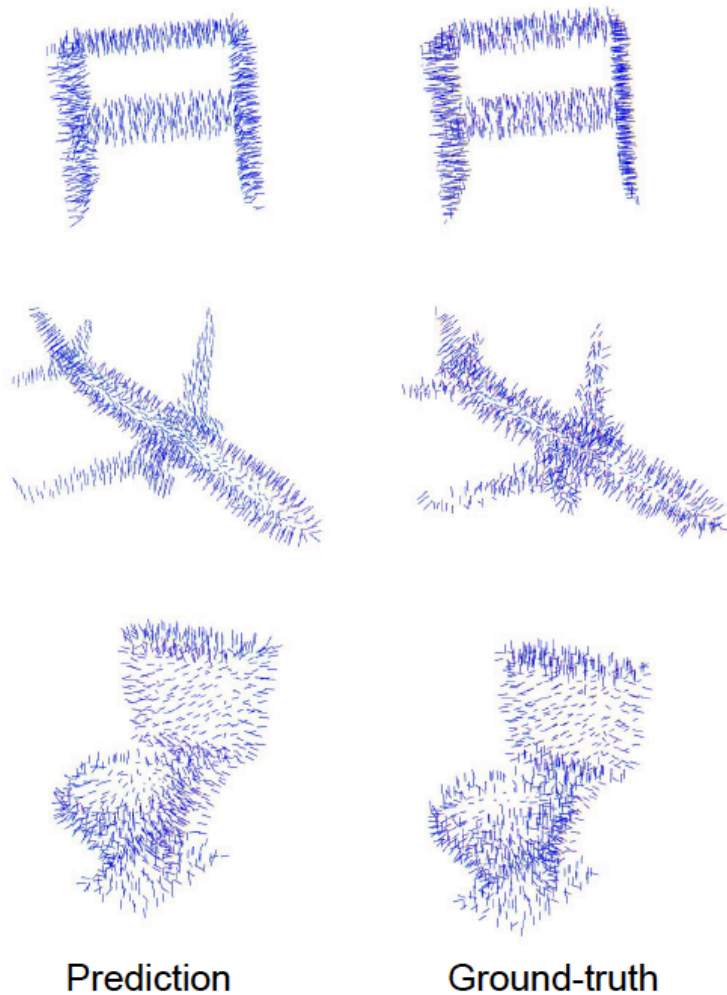


Figure 16. **PointNet normal reconstruction results.** In this figure, we show the reconstructed normals for all the points in some sample point clouds and the ground-truth normals computed on the mesh.

Qi et al 17

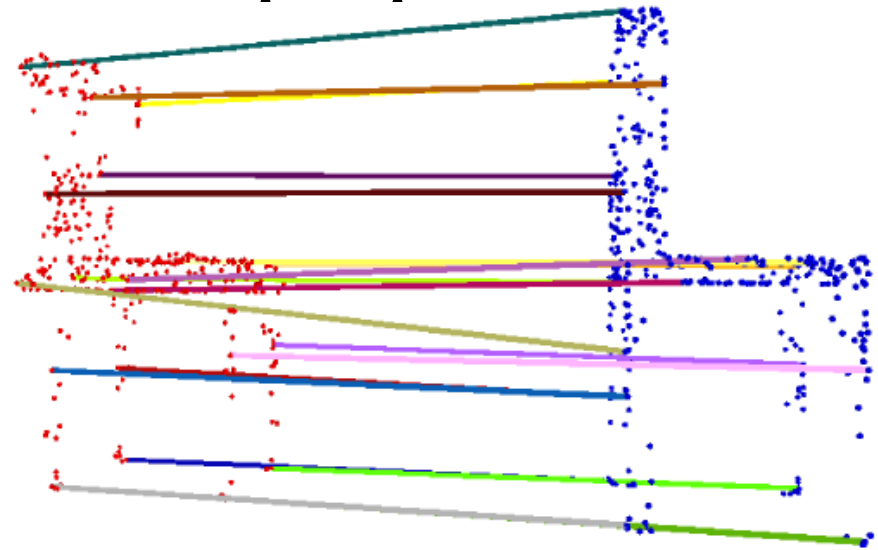


Figure 13. **Shape correspondence between two chairs.** For the clarity of the visualization, we only show 20 randomly picked correspondence pairs.

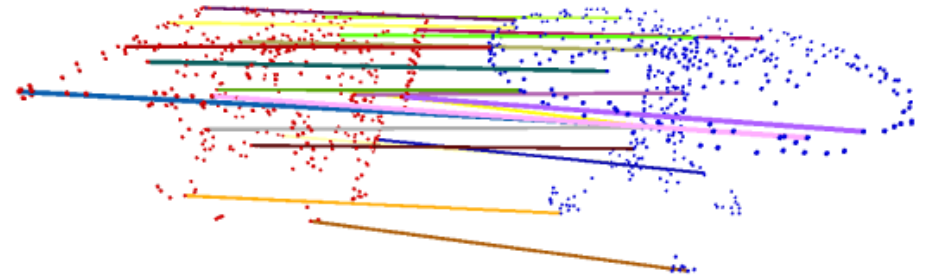


Figure 14. **Shape correspondence between two tables.** For the clarity of the visualization, we only show 20 randomly picked correspondence pairs.

This is a general summarization procedure

- Point clouds aren't just 3D points
- Examples:
 - (x, y, z, r, g, b)
 - $(x, y, z, \text{feature vector})$
 - feature vectors of a batch
 - useful idea in adversarial learning?
 - center positions, params, weights of each metaball

Pointnet++: Further tricks

- Clustering points is permutation invariant
 - so one could build clusters from a point cloud, then describe those

Notes and queries

- Claim: the set of points that represent the set is sparse
 - At most K points participate, so if true if $K < n$
 - not true otherwise
- Q: Assume we formed $\gamma \left(\underset{x_i \in S}{MAX} \{h(x_i)\} \right)$
 - and y_i close to x_i
 - what is $\gamma(\max\{h(y_i)\})$ like?
 - likely controlled by learning procedure

CvxNet

- Represent objects as union of convexes
 - convexes == polyhedra, with smoothed indicator function
- Important:
 - convexes are intersections of half planes, rather than (say) vertices, meshes
 - this is a CSG representation
 - union of intersections

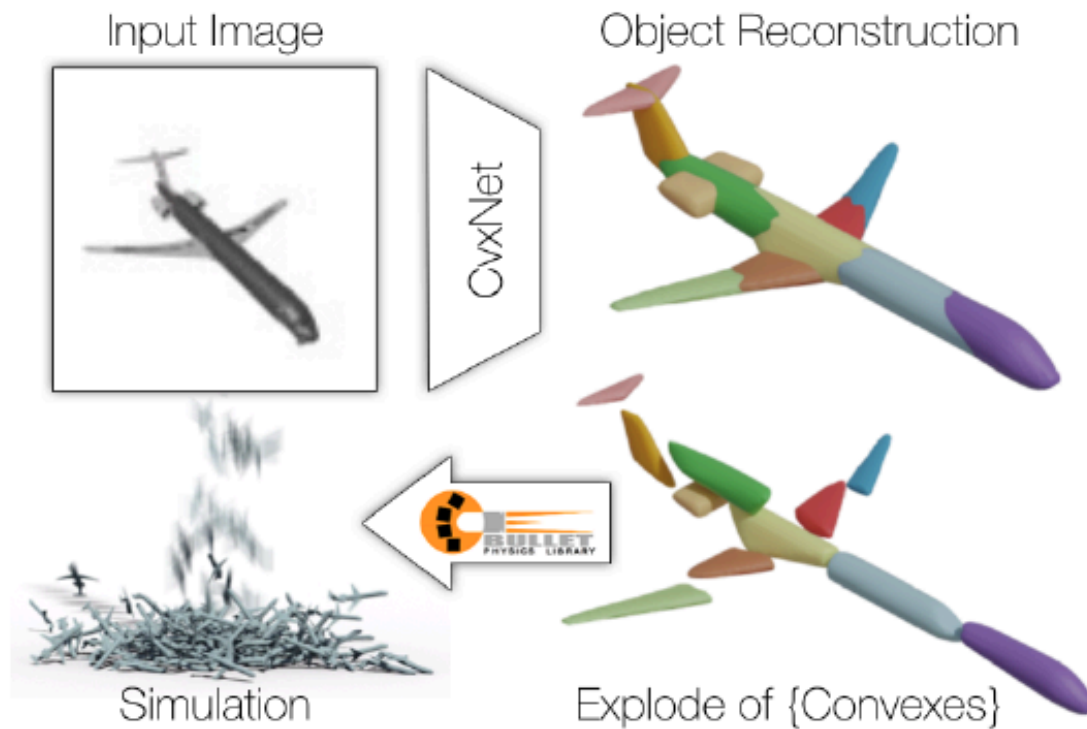


Figure 1. Our method reconstruct a 3D object from an input image as a collection of convex hulls, and we visualize the explode of these convexes. Notably, CvxNet outputs polygonal mesh representations of convex polytopes *without* requiring the execution of computationally expensive iso-surfacing (e.g. Marching Cubes). This means the representation outputted by CvxNet can then be readily used for physics simulation [17], as well as many other downstream applications that consume polygonal meshes.

Convex sets

- A set of points C is convex if

$$\text{for } x_1, x_2 \in C \text{ and } 0 \leq t \leq 1, tx_1 + (1 - t)x_2 \in C$$

- A hyperplane H is a supporting hyperplane of C if:
 - at least one point of C lies on H
 - all of C lies on one side of H
- Every point on C has at least one supporting hyperplane passing through it - but there can be more
- C is the intersection of the half-spaces defined by supporting hyperplanes

Convex sets

- Can write \mathcal{C} as

$$\mathcal{C} = \{x \mid \pi_i(x) + c_i \geq 0\}$$

- where $\pi_i(x) + c_i$ is the equation of the i 'th supporting hyperplane on x
- deliberate vagueness about indexing here
 - there may be an infinite set of supporting hyperplanes
- Confine to finite sets of supporting hyperplanes, so

$$\mathcal{C} = \{\mathbf{x} \mid \mathcal{M}\mathbf{x} + \mathbf{c} \geq 0\}$$

- \mathcal{M} a matrix, \mathbf{c} a vector
- Such convex sets are not necessarily bounded

A construction....

- Given a set of hyperplanes, how do we know it makes a convex set?

A construction....

- Assume C is not empty, then it contains some point
 - so there is some point x such that

$$\mathcal{M}x + \mathbf{c} \geq 0$$

- Notice that any non-negative combination of inequalities is also valid for every point in the set
 - Picture can help here;
 - but basically, these inequalities are + at every point in C
 - so a non-neg combination is also +
- If C is empty, then some combination of inequalities should make this obvious

A construction....

- If C is empty, then some combination of inequalities should make this obvious
 - equivalently, I can derive a contradiction \implies inequality that can't be true for any point

- Now consider:

$$\mathbf{a} \text{ such that } \mathbf{a}^T \mathcal{M} = 0 \text{ and } \mathbf{a} \geq 0$$

- this represents a non-negative combination of inequalities, so should be non-negative for any point in C
- If, in addition, $\mathbf{a}^T \mathbf{c} < 0$, we have our contradiction

Contradiction

- If \mathbf{x} is in C , then

$$\mathbf{a}^T \mathcal{M}\mathbf{x} + \mathbf{a}^T \mathbf{c} \geq 0$$

- but by hypothesis

$$\mathbf{a}^T \mathcal{M}\mathbf{x} = 0$$

- so

$$\mathbf{a}^T \mathcal{M}\mathbf{x} + \mathbf{a}^T \mathbf{c} = \mathbf{a}^T \mathbf{c} < 0$$

Farkas' Lemma

- Either there exists an \mathbf{x} such that

$$\mathcal{M}\mathbf{x} + \mathbf{c} \geq 0$$

- or there exists an \mathbf{a} such that

$$\mathbf{a} \text{ such that } \mathbf{a}^T \mathcal{M} = 0 \text{ and } \mathbf{a} \geq 0 \text{ and } \mathbf{a}^T \mathbf{c} < 0$$

- (This is one of many forms of Farkas' Lemma; it turns up all over the place — eg linear programming in dual constructions; functional analysis as Hahn-Banach thrm)

Another form of Farkas' lemma

- Either a point x is in a convex set or there exists a hyperplane separating x from the convex set
 - i.e. such that the hyperplane is $+$ on convex set, and $-$ on point
- Prove as assignment in homework; straightforward

Two natural representations

- C represented by hyperplanes
 - finite set, so polyhedron or cone
 - may not be bounded
 - impressively easy to work with
- C as $\text{convex_hull}(p_1, \dots, p_k)$
 - bounded
- Issues:
 - passing from one to another can be tricky
 - in 2D relatively straightforward
 - in 3D do-able
 - in ND, hard

Easy constructions

- CvxNet represents a convex as a set of hyperplanes
 - fixed number

- To test:

- Point \mathbf{x} is inside a set

$$\min(\mathcal{M}\mathbf{x} + \mathbf{c}) \geq 0$$

- Point \mathbf{x} is outside a set

$$\min(\mathcal{M}\mathbf{x} + \mathbf{c}) < 0$$

- $\text{convex_hull}(p_1, \dots, p_n)$ inside a set

$$\min_i(\min(\mathcal{M}\mathbf{p}_i + \mathbf{c})) \geq 0$$

i.e. if every point is in, then hull is in

What CvxNet does...

- We have a set of sample points inside and outside object
- Object is a union of a fixed number of convexes
 - some smoothing of the indicator function with sigmoid; largely ignore
- Choose these convexes so that
 - every point that should be inside is inside some convex
 - every point that should be outside is outside every convex
 - every convex accounts for at least one sample point

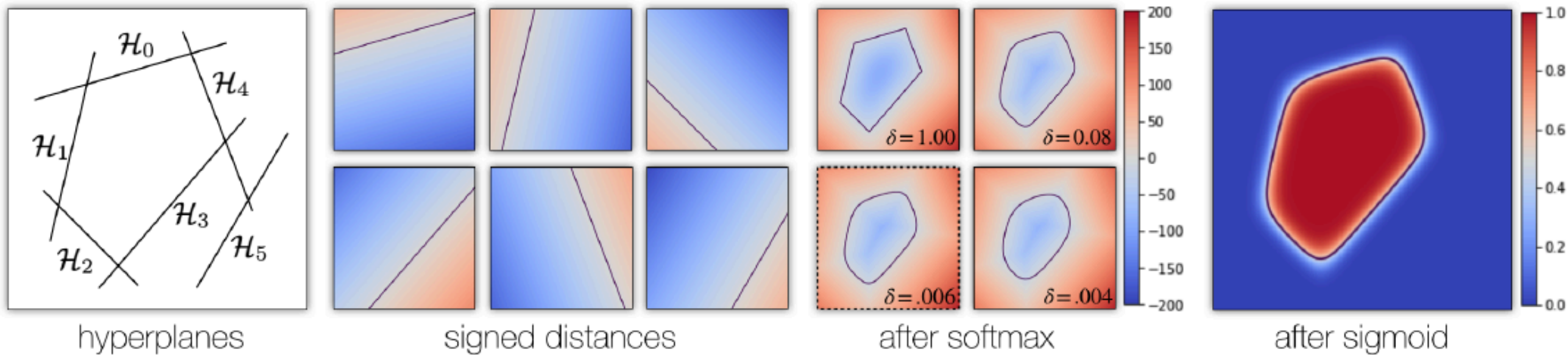


Figure 2. **From {hyperplanes} to occupancy** – A collection of hyperplane parameters for an image specifies the indicator *function* of a convex. The soft-max allows gradients to propagate through all hyperplanes and allows for the generation of *smooth* convex, while the sigmoid parameter controls the *slope* of the transition in the generated indicator – note that our soft-max function is a LogSumExp.

Deng et al 20

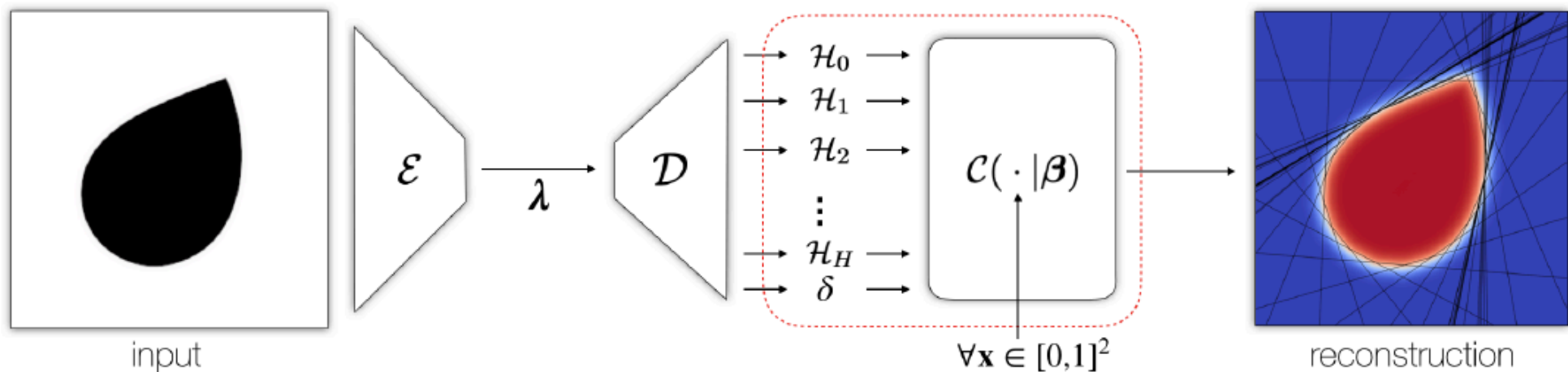


Figure 3. **Convex auto-encoder** – The encoder \mathcal{E} creates a low dimensional latent vector representation λ , decoded into a collection of hyperplanes by the decoder \mathcal{D} . The training loss involves reconstructing the value of the input image at random pixels \mathbf{x} .

?!?!?!?!?!#@%?!?!?!?!?

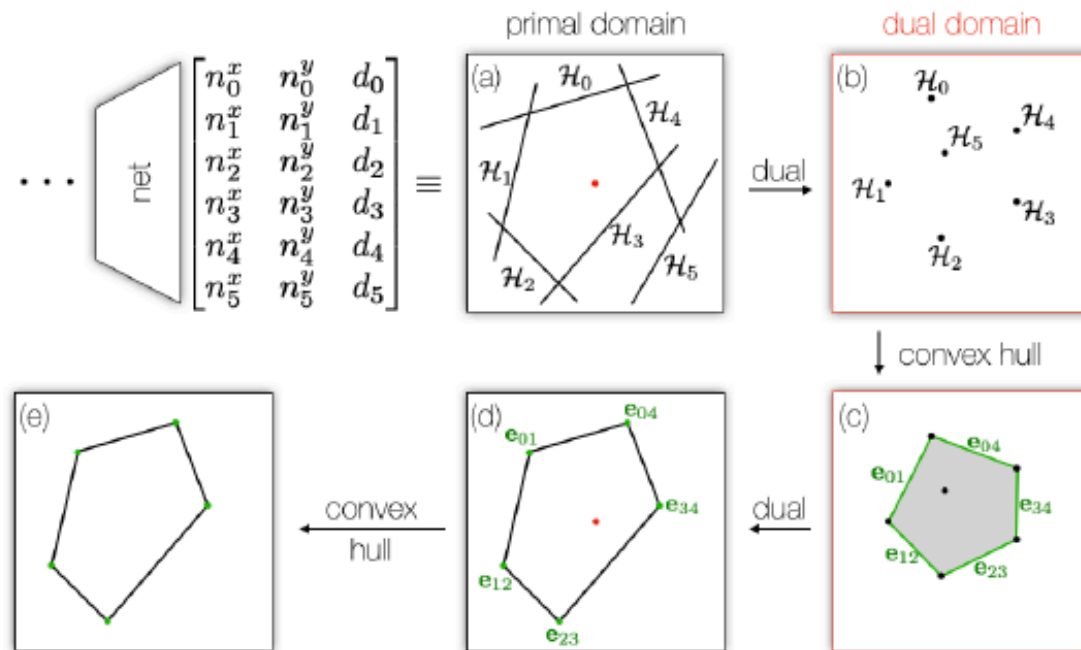


Figure 6. **From {hyperplanes} to polygonal meshes** – The polygonal mesh corresponding to a set of hyperplanes (a) can be computed by transforming planes into points via a duality transform (b), the computation of a convex hull (c), a second duality transform (d), and a final convex hull execution (e). The output of this operation is a *polygonal mesh*. Note this operation is efficient, output sensitive, and, most importantly does not suffer the curse of dimensionality. Note that, for illustration purposes, the duality coordinates in this figure are fictitious.

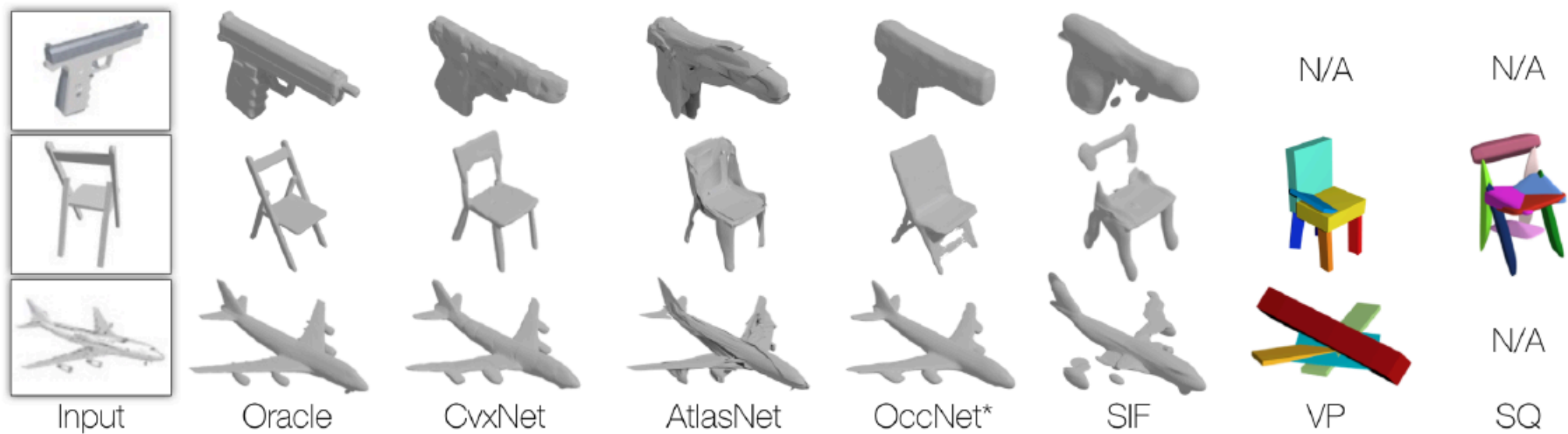


Figure 12. **ShapeNet/Multi** – Qualitative comparisons to SIF [21], AtlasNet [26], OccNet [44], VP [68] and SQ [50]; on RGB Input, while VP uses voxelized, and SQ uses a point-cloud input. (* Note that the OccNet [44] results are post-processed with smoothing).

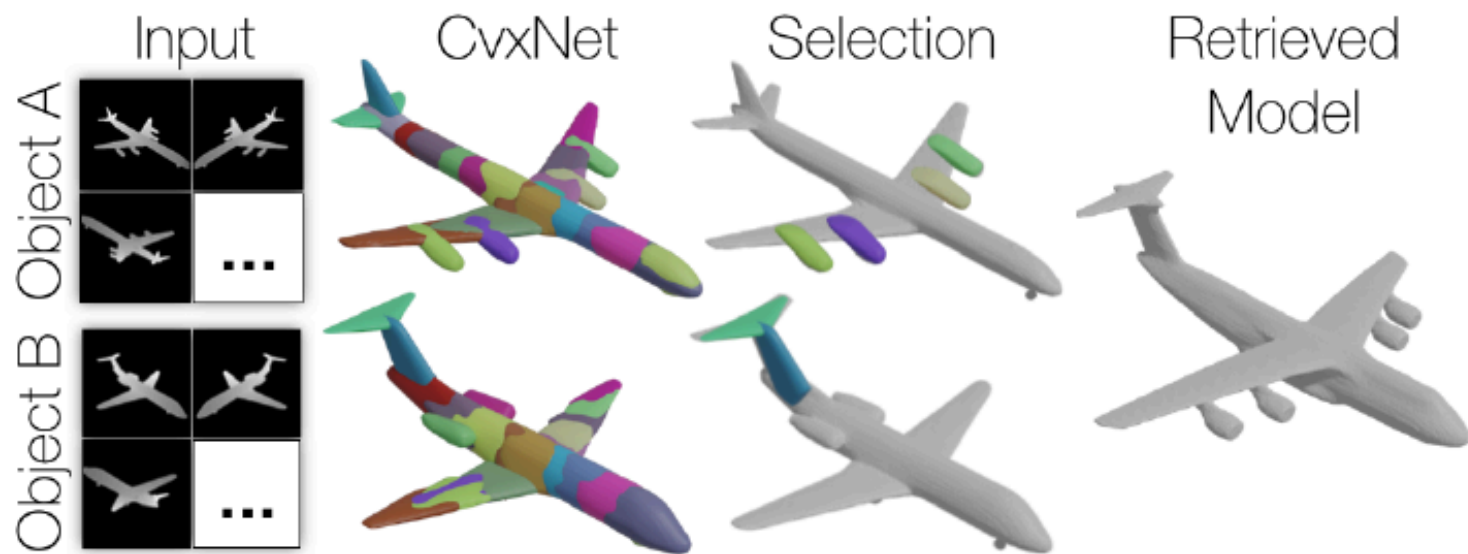


Figure 10. **Part based retrieval** – Two inputs (left) are first encoded into our CvxNet representation (middle-left), from which a user can select a subset of parts (middle-right). We then use the concatenated latent code as an (incomplete) geometric lookup function, and retrieve the closest decomposition in the training database (right).

N+Q

- Very neat fitting results
- Some cases will work badly (I'll draw)
- Samples are a fantastically inefficient geometric rep'n (to follow)
- Part claim is weird
 - why bother? why not use partnet style rep'n on params of convex
- Why not use other CSG constructions?
 - this is union of intersections
 - but we could do differences - eg rooms

Harder

- A convex set D represented by \mathcal{M} , d is outside C
 - equivalently, there is no point inside D and C
 - equivalently, there exists \mathbf{a} such that

$$\mathbf{a}^T \begin{bmatrix} \mathcal{M} \\ \mathcal{N} \end{bmatrix} = 0 \text{ and } \mathbf{a}^T \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} < 0 \text{ and } \mathbf{a} \geq 0$$

- You could write this as a loss
 - first term: minimize $\mathbf{a}^T \mathbf{a}$
 - second term introduce slacks, etc to get hinge loss
 - third term introduce slacks, etc. to get hinge loss

Slacks

- We want

$$\mathbf{a}^T \mathbf{w} < 0$$

- Choose a scale to get (for xi positive)

$$\mathbf{a}^T \mathbf{w} \leq -1 + \xi$$

- Or

$$\mathbf{a}^T \mathbf{w} + 1 \leq \xi$$

- So minimize

$$\xi = \max(\mathbf{a}^T \mathbf{w} + 1, 0)$$

Notice

- This construction doesn't penalize volume
 - so size of loss is not proportional to volume of intersection
 - likely quite hard to do

Constructions:

- How do we know two convex sets do not intersect
 - FM elimination