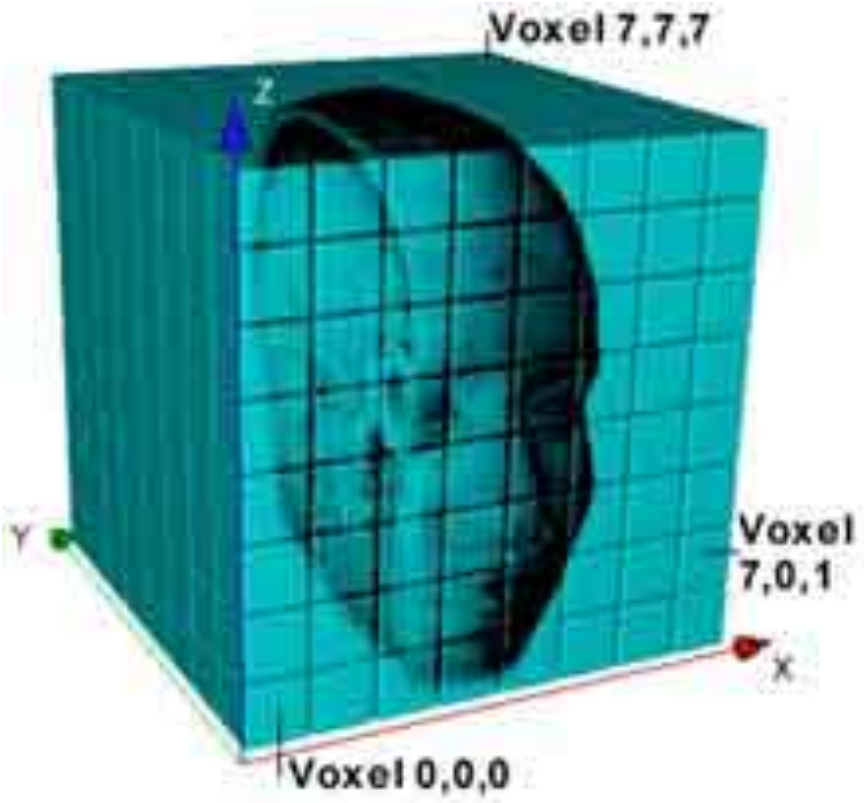
Constructive Solid Geometry and Procedural Modeling

Sai-Kit Yeung

ISTD, SUTD

Previous Lecture: Solid Modeling

- Represent solid interiors of objects
- Voxels
- Octrees
- Tetrahedra
- Distance Fields



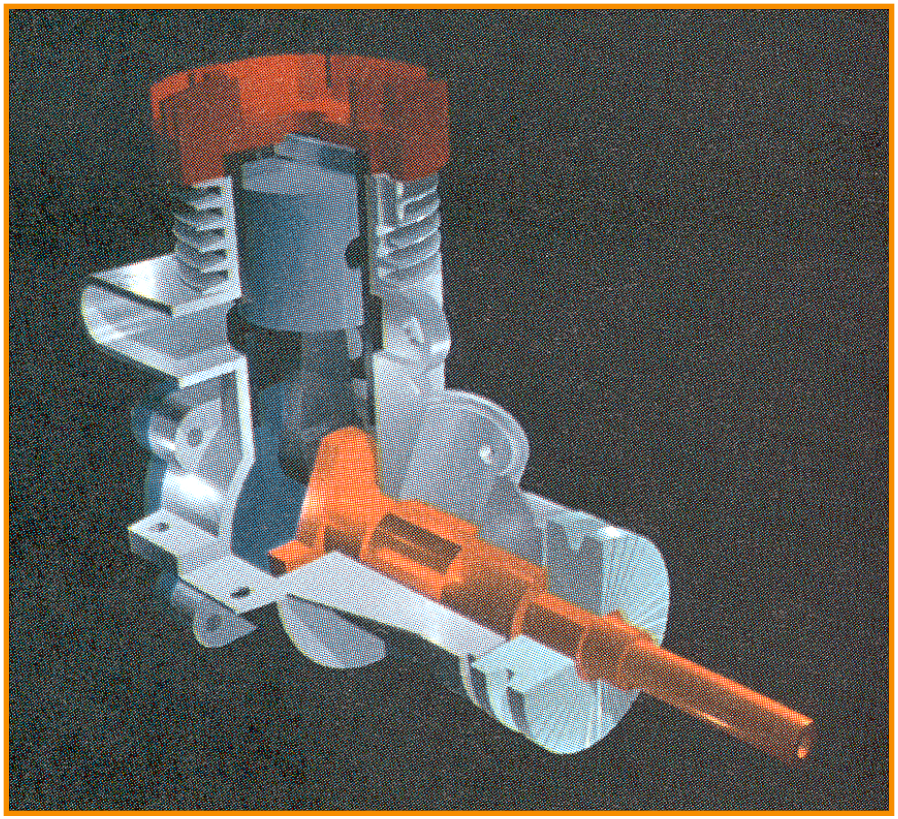
www.volumegraphics.com

The Plan For Today

- Constructive Solid Geometry (CSG)
- Procedural Geometry Modeling
- OpenSCAD

Constructive Solid Geometry (CSG) in CAD

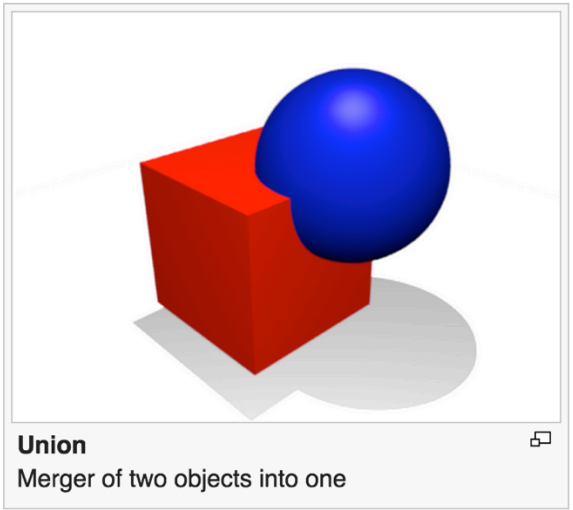
- Interactive modeling programs
  - Intuitive way to design objects



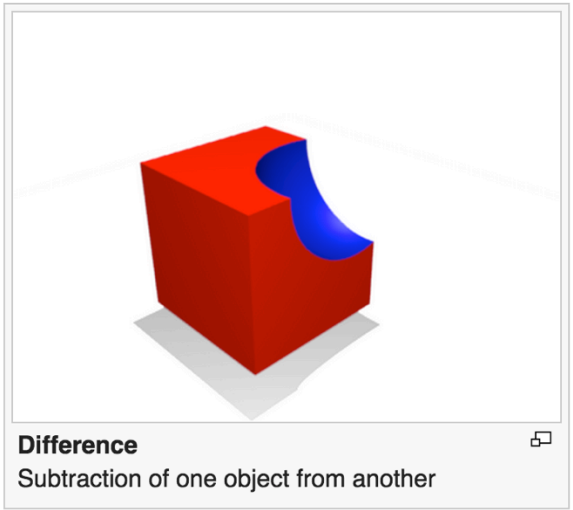
H&B Figure 9.9

Constructive Solid Geometry (CSG)

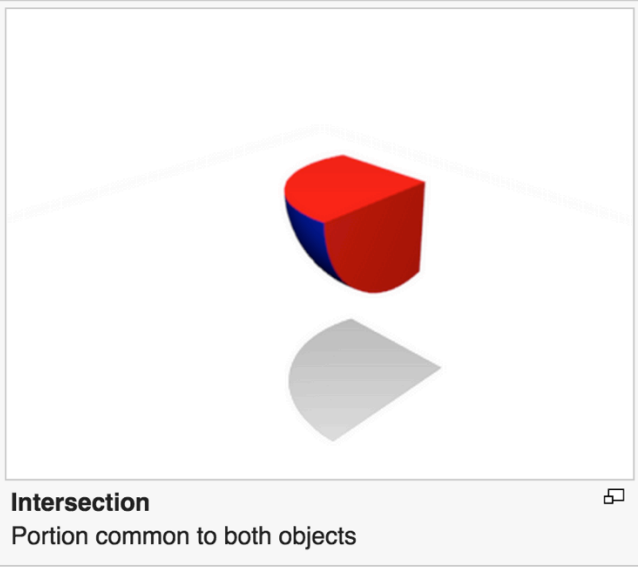
- Boolean operations
  - Union
  - Intersection
  - Difference



Union
Merger of two objects into one



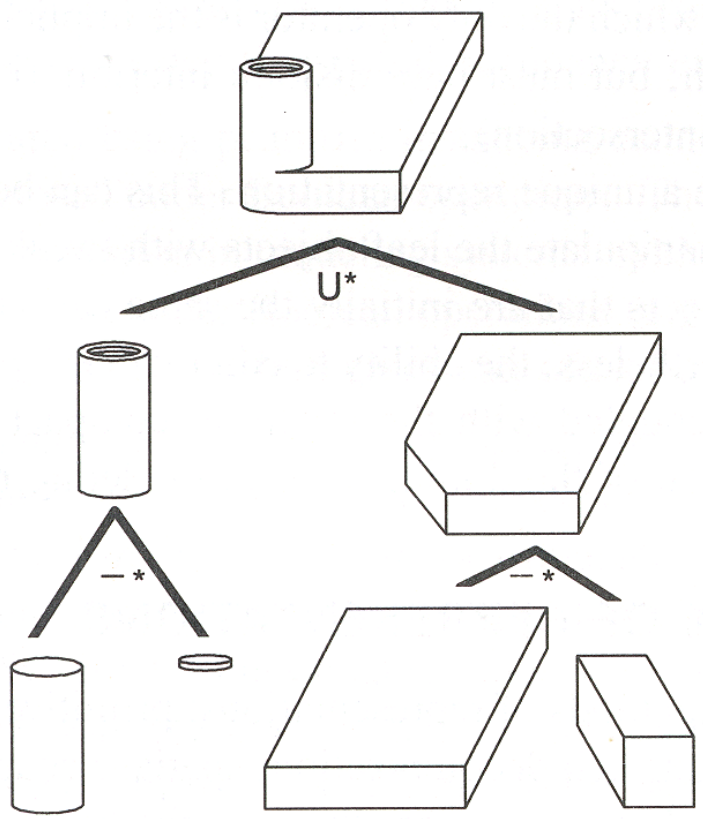
Difference
Subtraction of one object from another



Intersection
Portion common to both objects

Constructive Solid Geometry (CSG)

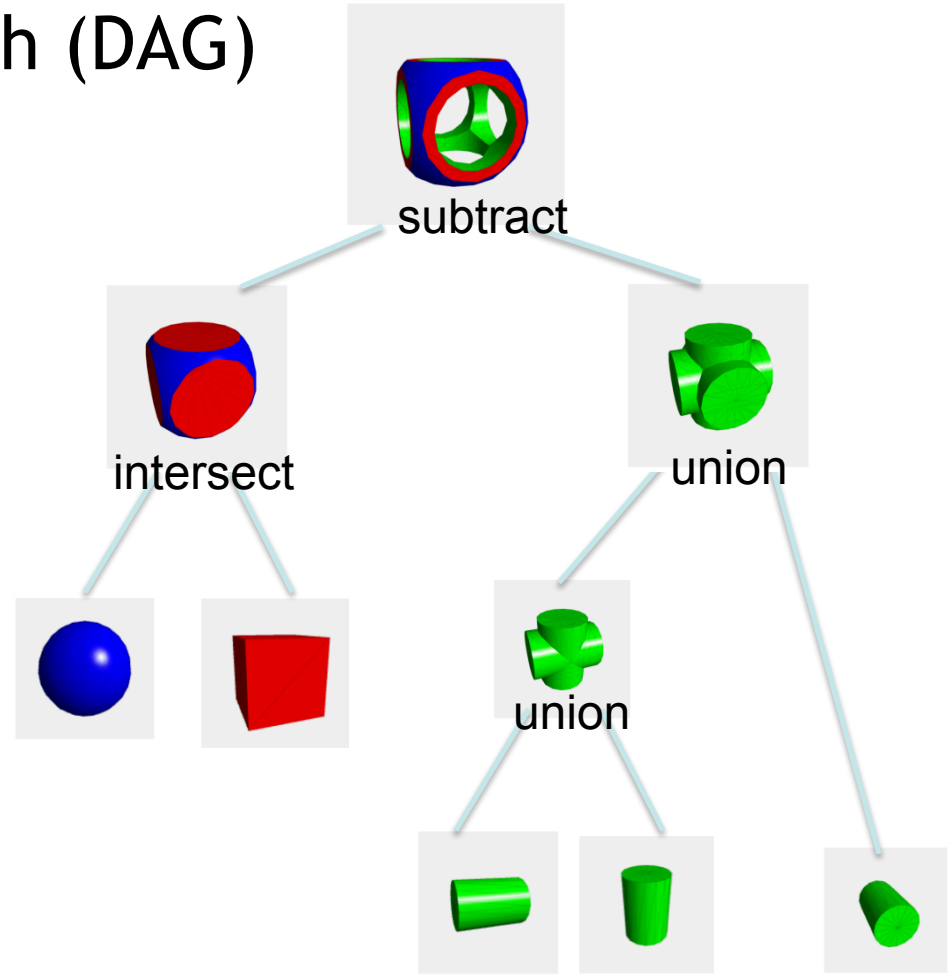
- Represent solid object as hierarchy of Boolean operations
- Objects are represented implicitly with a tree structure



FvDFH Figure 12.27

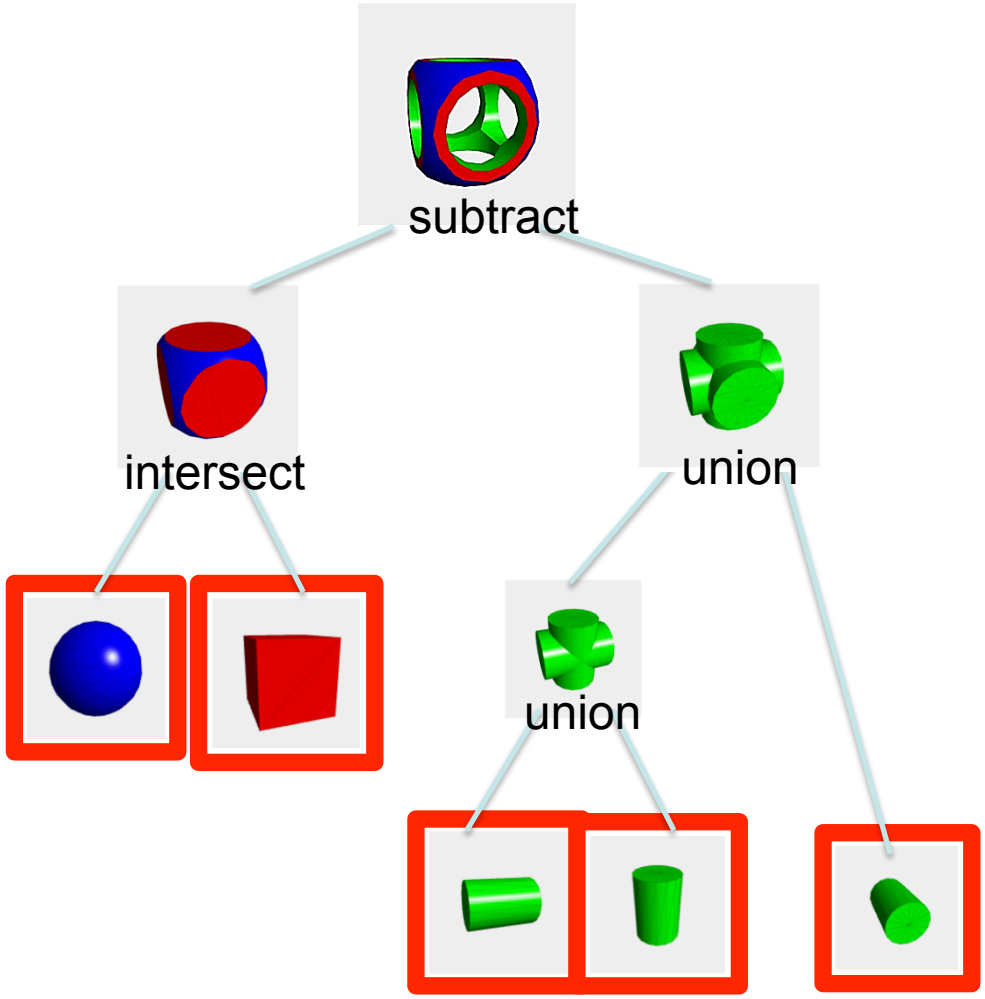
CSG Data Structure

- Binary Tree
- Directed Acyclic Graph (DAG)



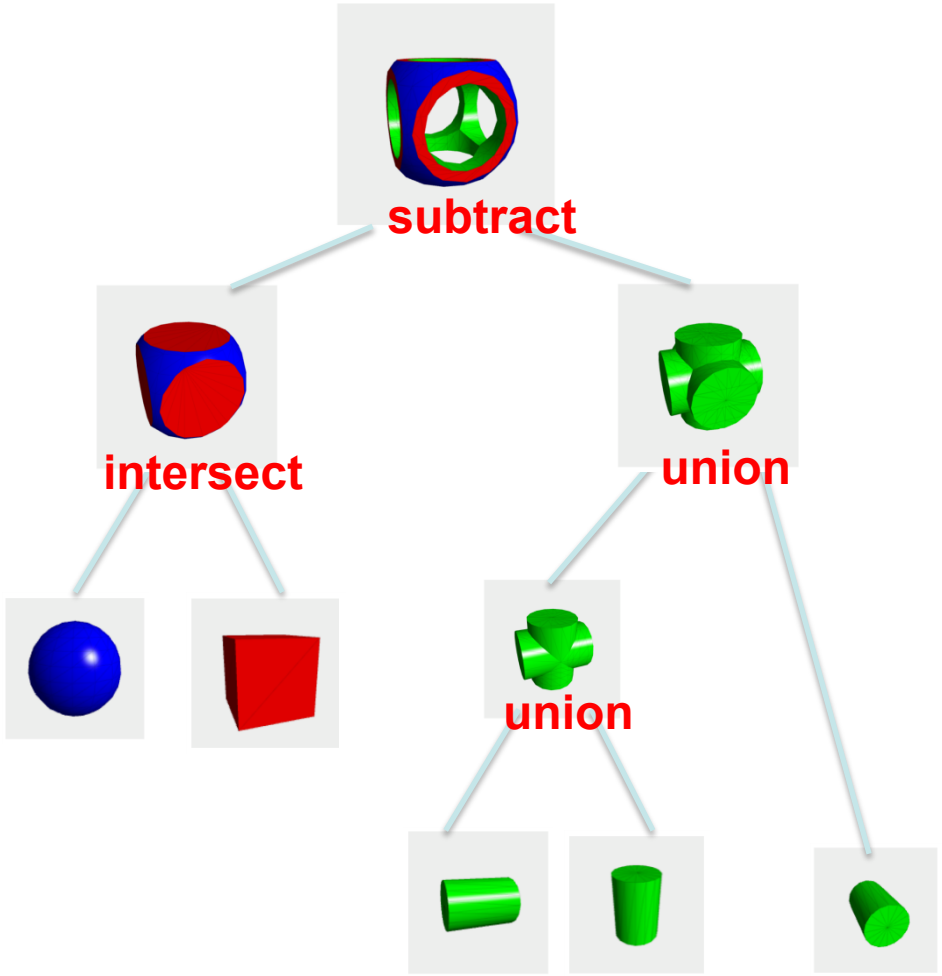
Leaves: CSG Primitives

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

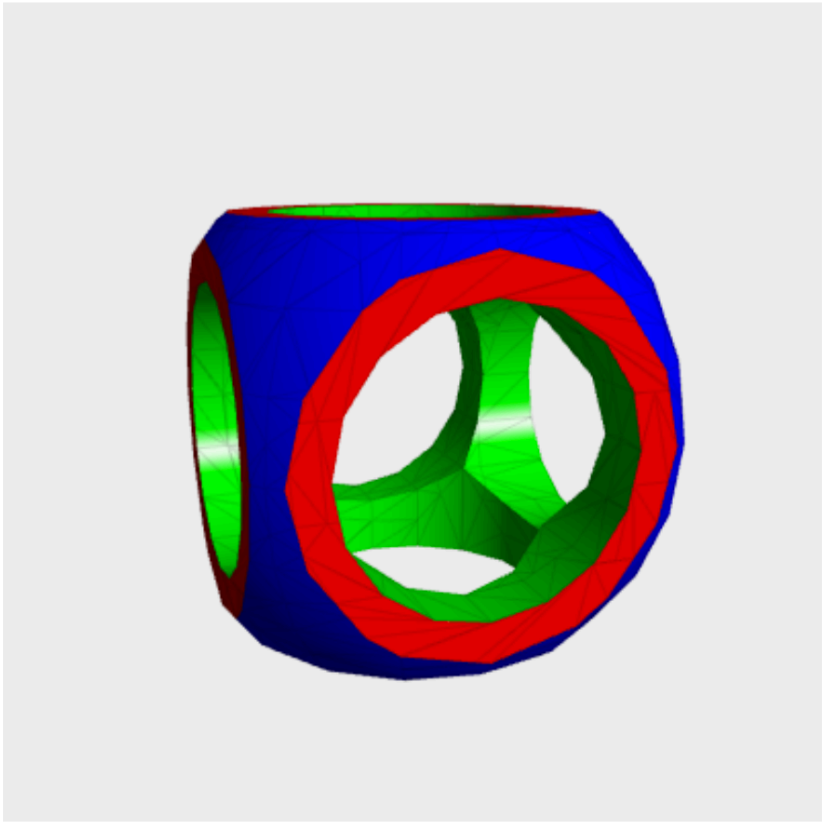


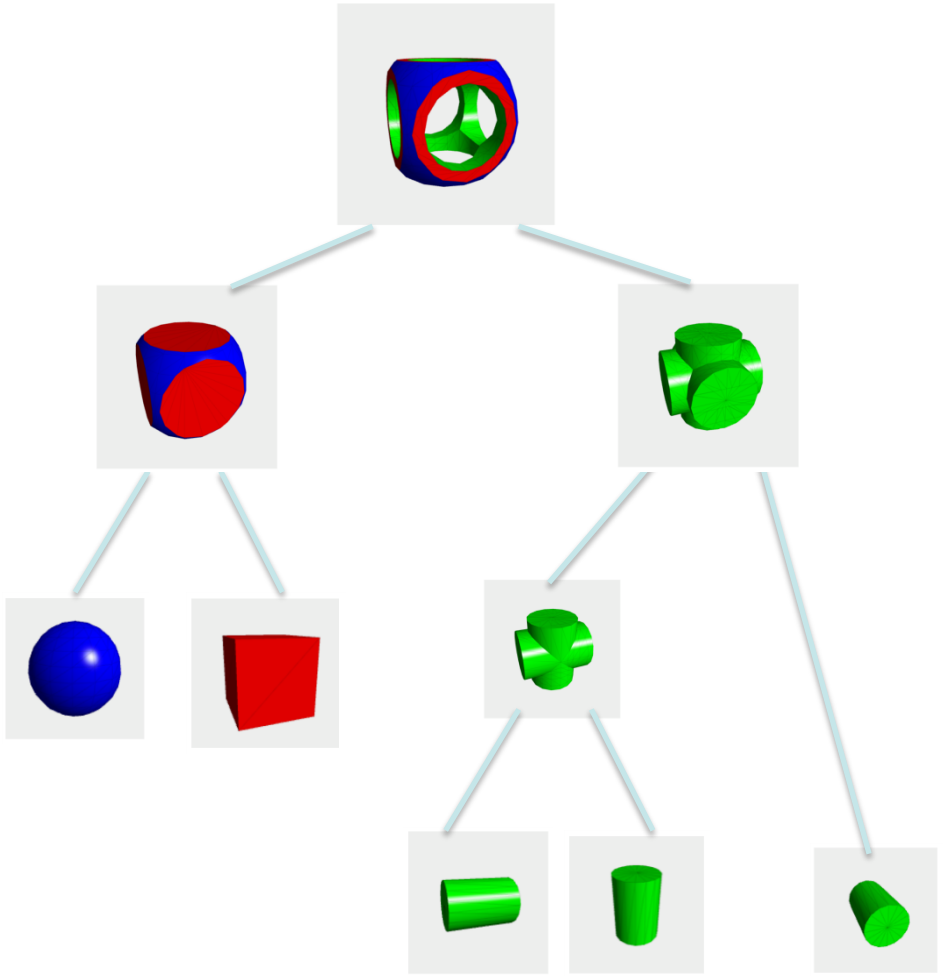
Internal Nodes

- Boolean Operations
  - Union
  - Intersection
  - Difference
- Rigid Transformations
  - Scale
  - Translation
  - Rotation
  - Shear



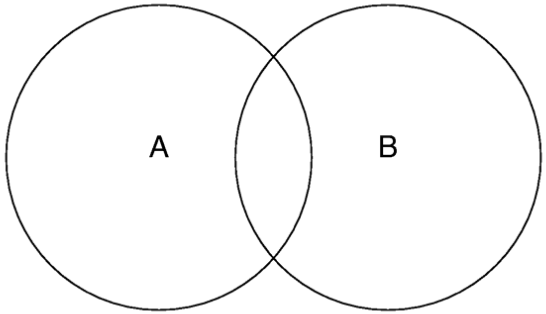
Root: The Final Object



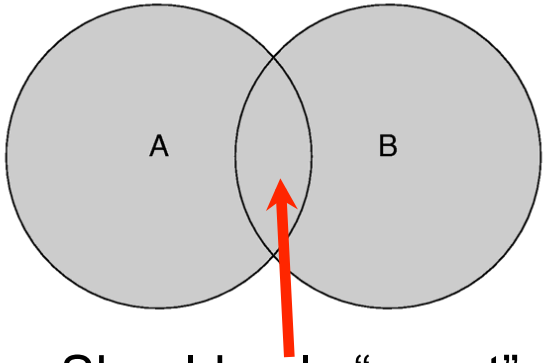


Recap: Booleans for Solids

Given overlapping shapes A and B:

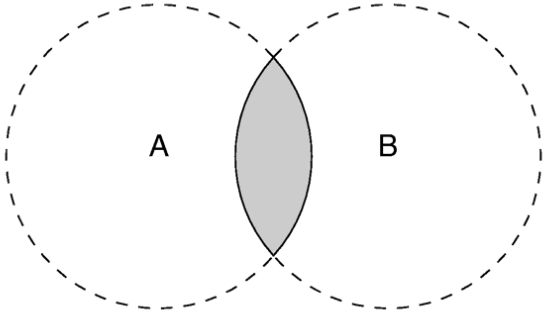


Union

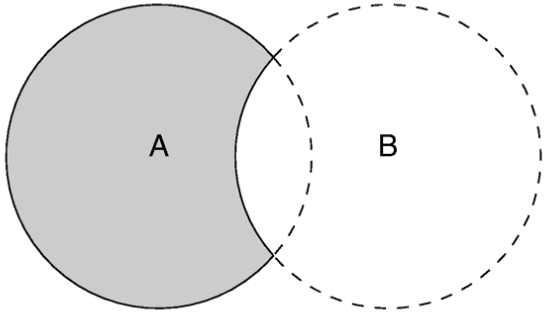


Should only “count” overlap region once!

Intersection

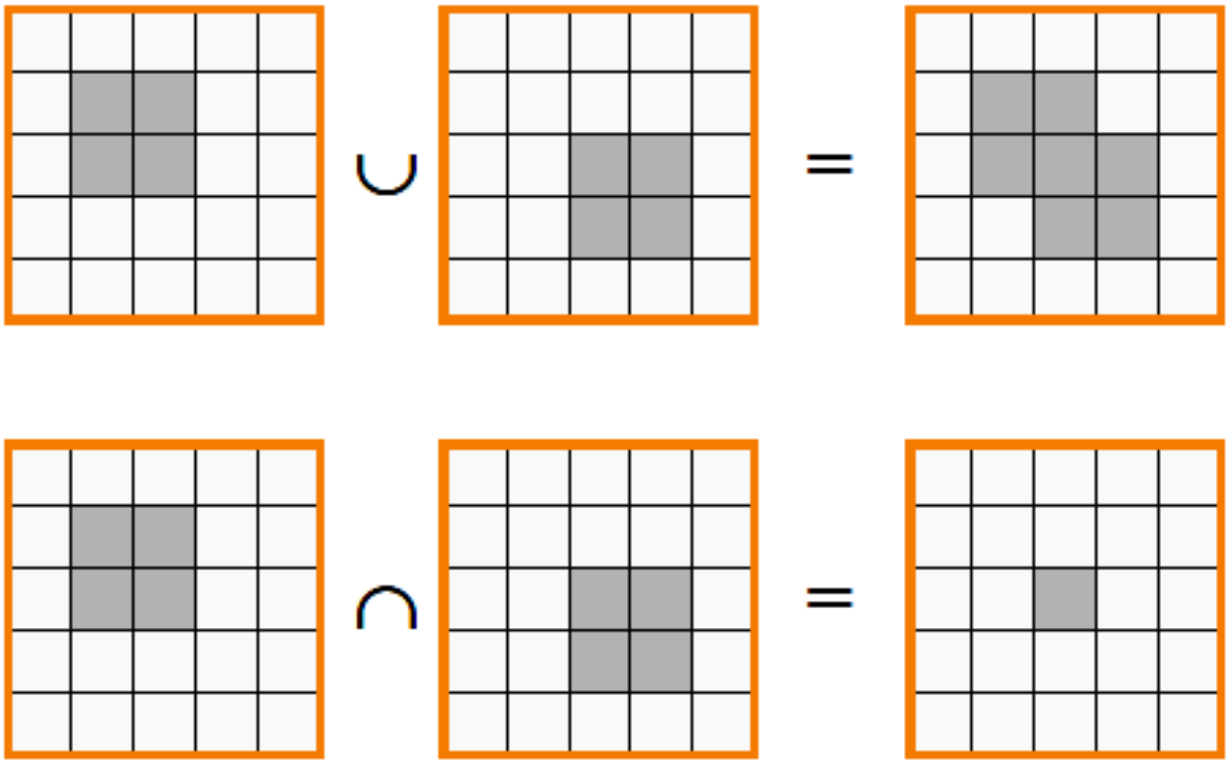


Subtraction



How Can We Implement Boolean Operations?

- Use voxels/octrees/ADFs
  - We can convert from surface to voxels and back
  - Compare objects voxel by voxel

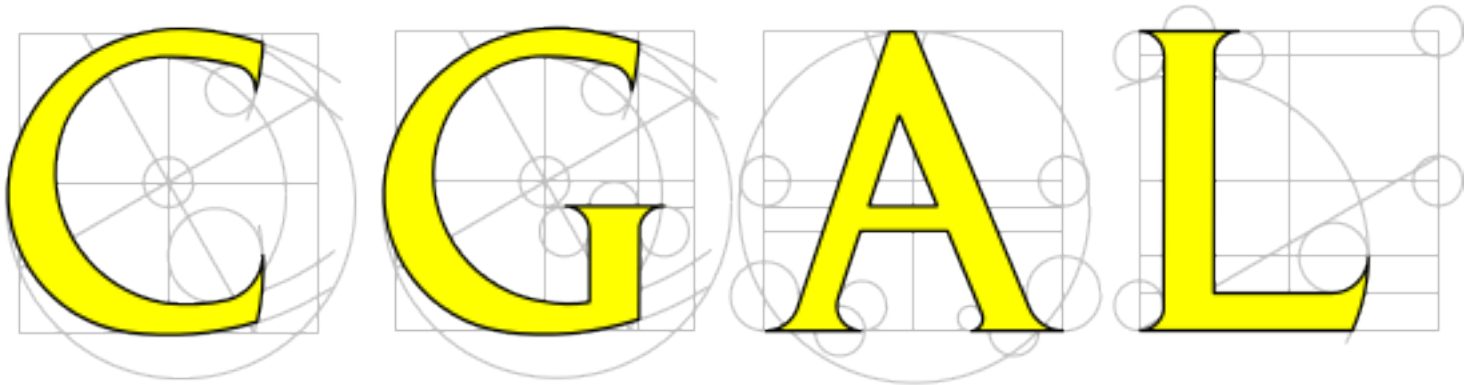


How Can We Implement Boolean Operations?

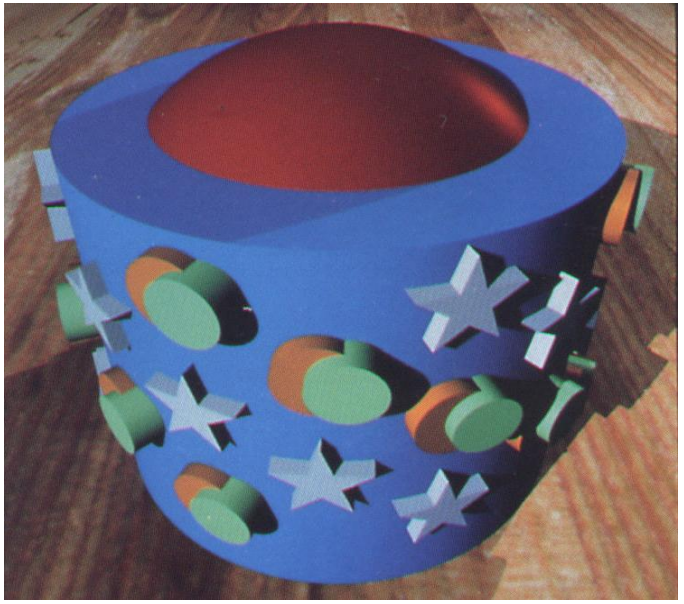
- We will not be asking you to do this
- Commercial libraries
  - e.g., Parasolid
- Open source libraries
  - e.g., CGAL

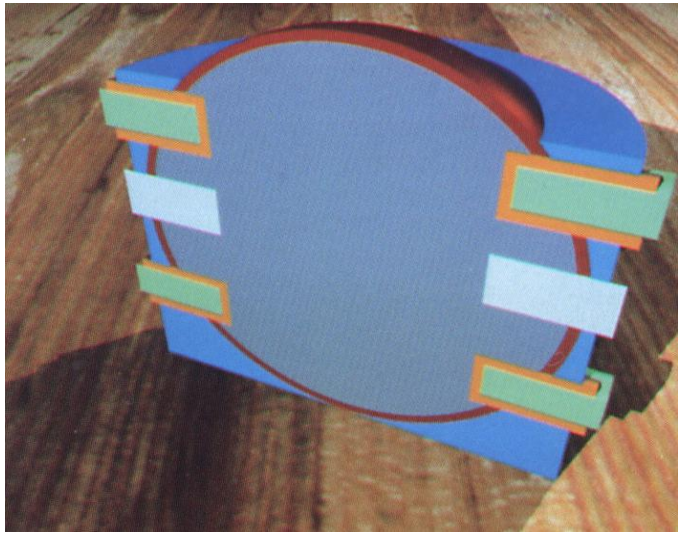
Computational Geometry Algorithms Library

- CGAL is open source
- Data structures and algorithms
  - Triangulations
  - Voronoi diagrams
  - **Boolean operations**
  - Mesh generation
  - Geometry processing
  - Search structures, …



CSG Examples





Demo

- http://evanw.github.io/csg.js/

The Plan For Today

- Constructive Solid Geometry (CSG)
- Procedural Modeling
- OpenSCAD

Procedural Modeling

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

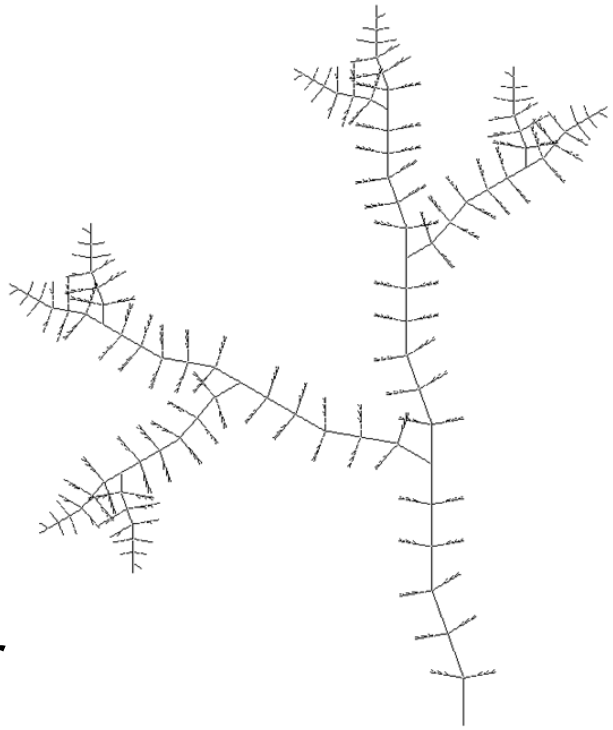
Formal Grammars and Languages

- A finite set of **nonterminal symbols: {S, A, B}**
- A finite set of **terminal symbols: {a, b}**
- A finite set of **production rules: S → AB**
- 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

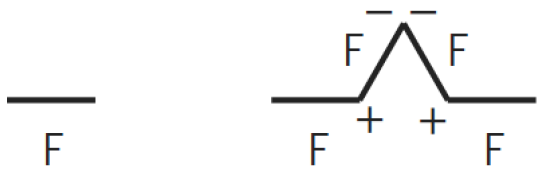


L-systems

- String rewriting system
- Begin with a set of “productions” (replacement rule) and a “seed” axiom
- In parallel, all matching productions are replaced with their right-hand sides, e.g.,
  - Rules:
    - B → ACA
    - A → B
  - Axiom: AA
  - Sequence: AA, BB, ACAACA, BCBBCB, …
- Strings can be converted to graphic representation via interpretation as turtle graphics commands

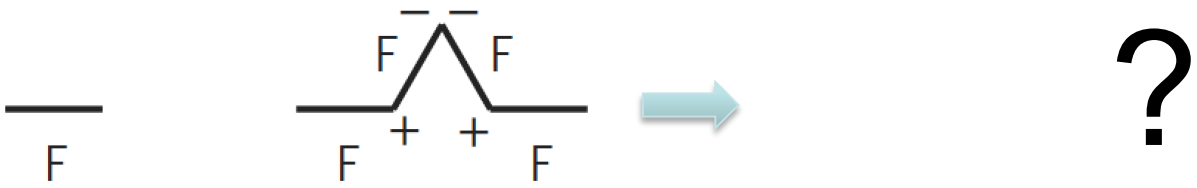
Turtle Commands

- F: move forward (leave a trail)
- +: rotate the turtle counter-clockwise
- -: rotate the turtle clockwise
- For example, a production might be:
  - F → F + F - - F + F
  - Setting rotation to 60^{o}

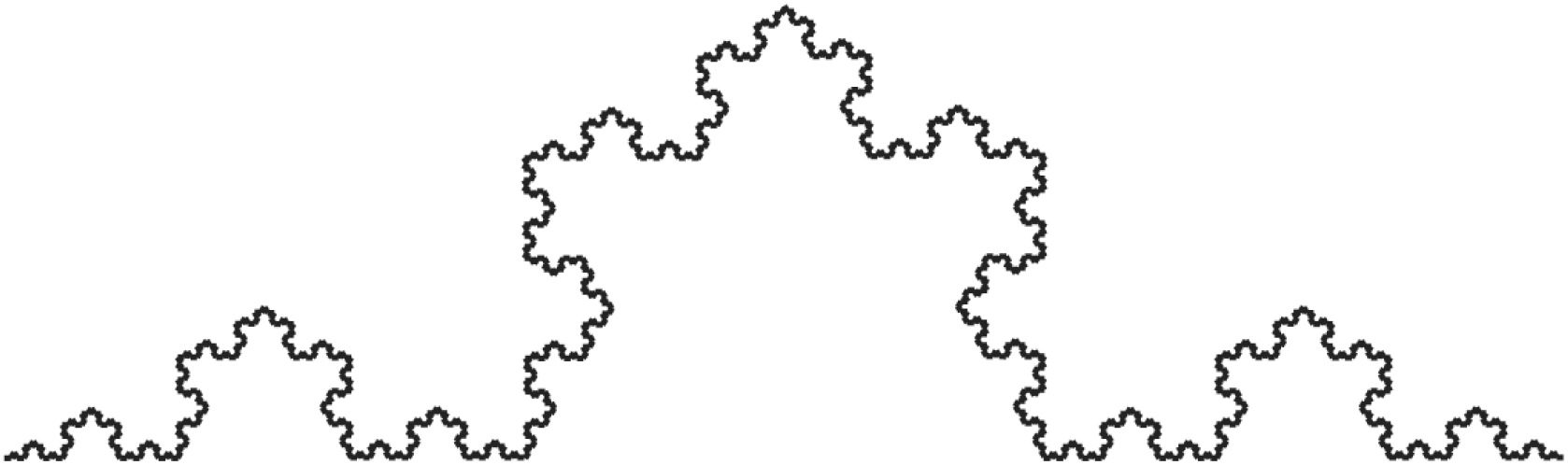


Turtle Commands

- Exercise



Turtle Commands



Koch snowflake curve

L-system Example 1: Algae

- **nonterminals** : A B
- **terminals** : none
- **start** : A
- **rules** : (A → AB), (B → A)

n = 0 : A

n = 1 : AB

n = 2 : ABA

n = 3 : ABAAB

n = 4 : ABAABABA

n = 5 : ABAABABAABAAB

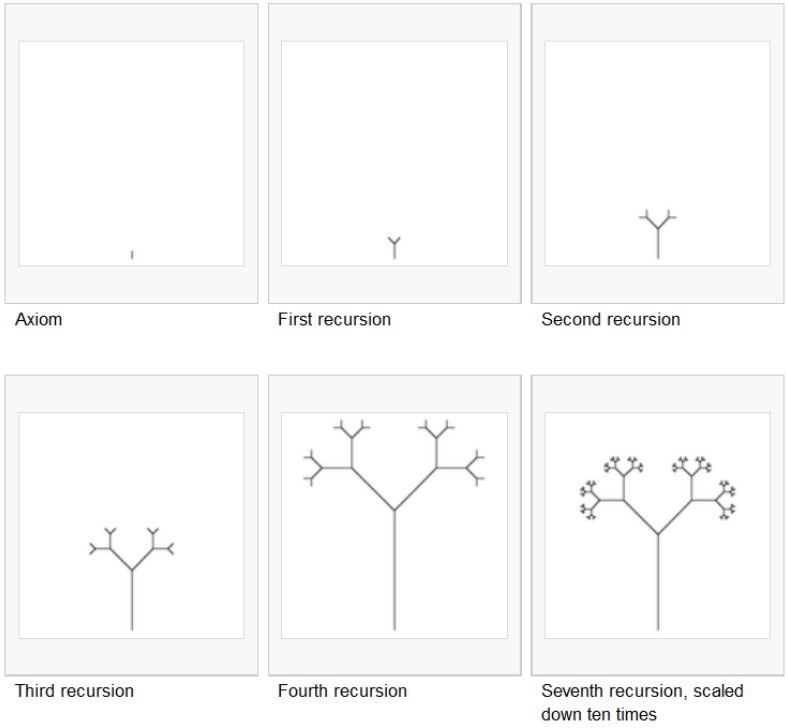
L-system Example 2

- **nonterminals** : 0, 1
- **terminals** : [,]
- **start** : 0
- **rules** : (1 → 11), (0 → 1[0]0)

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 2

- 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



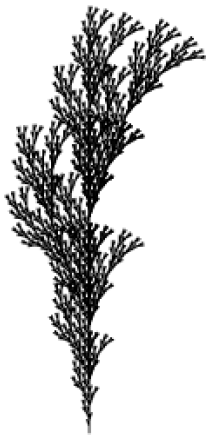
L-system Example 3: Fractal Plant

- **nonterminals** : X, F
- **terminals** : + - [ ]
- **start** : X
- **rules** : (X → F-[[X]+X]+F[+FX]-X), (F → FF)

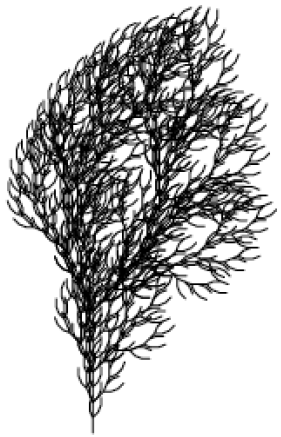
L-Systems Examples

- Tree examples

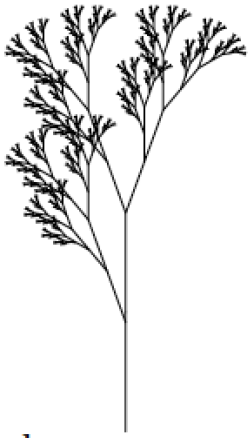
a
n=5, δ=25.7°
F
F→F[+F]F[-F]F



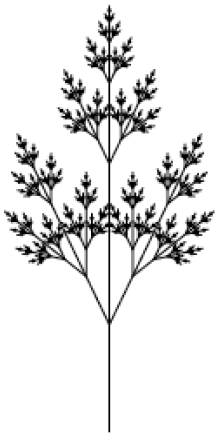
b
n=5, δ=20°
F
F →F[+F]F[-F][F]



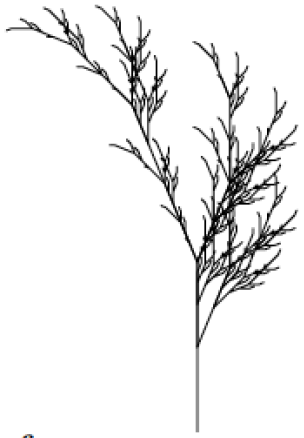
c
n=4, δ=22.5°
F
F→FF-[-F+F+F]+
[+F-F-F]



d
n=7, δ=20°
X
X →F[+X]F[-X]+X
F →FF



e
n=7, δ=25.7°
X
X→F[+X][-X]FX
F→FF



f
n=5, δ=22.5°
X
X→F-[[X]+X]+F[+FX]-X
F→FF

L-Systems Examples

Types of L-Systems

- *Deterministic*: If there is exactly one production for each symbol

  $0 \rightarrow 1[0]0$

- *Stochastic*: If there are several, and each is chosen with a certain probability during each iteration

  $0\ (0.5) \rightarrow 1[0]0$

  $0\ (0.5) \rightarrow 0$

Types of L-Systems

- *Context-free:* production rules refer only to an individual symbol
- *Context-sensitive:* the production rules apply to a particular symbol only if the symbol has certain neighbours

$$
\begin{aligned}
S &\rightarrow aSBC \\
S &\rightarrow aBC \\
CB &\rightarrow HB \\
HB &\rightarrow HC \\
HC &\rightarrow BC \\
aB &\rightarrow ab \\
bB &\rightarrow bb \\
bC &\rightarrow bc \\
cC &\rightarrow cc
\end{aligned}
$$

Types of L-Systems

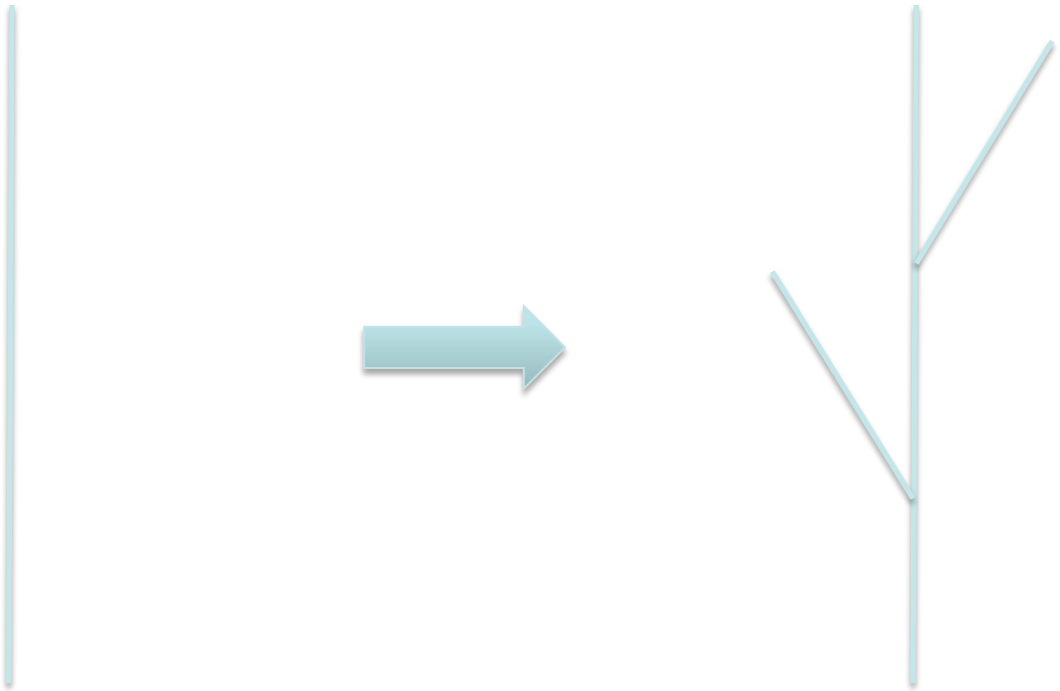
- *Nonparametric grammars:* no parameters associated with symbols
- *Parametric grammars:* symbols can have parameters
  - Parameters used in conditional rules
  - Production rules modify parameters
  - $A(x,y) : x = 0 \rightarrow A(1, y+1)B(2,3)$

Bracketed L-Systems

- Save and restore state (for branches)
- Useful for creating plant
- Left bracket symbol “[“ records its state (position and orientation)
- Right bracket symbol “]” instantly teleports the turtle to its most recently stored state
- We saw in example 2 and 3 before

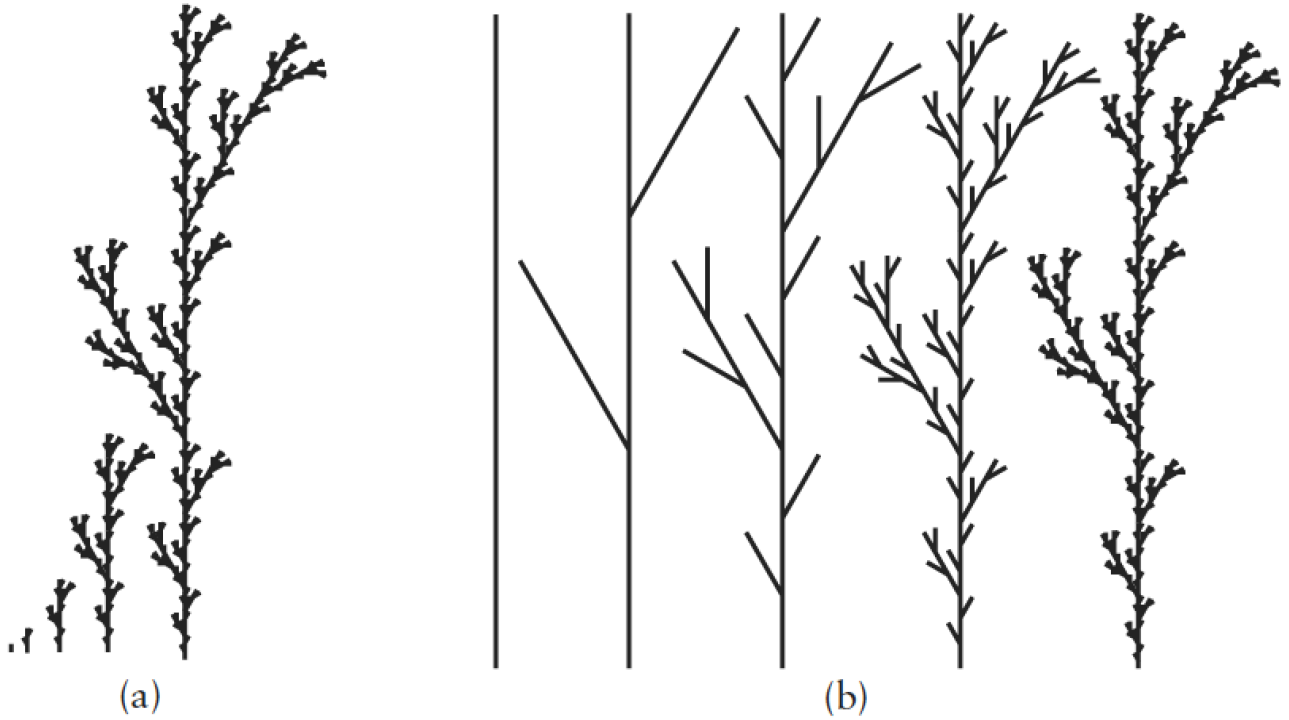
Bracketed L-Systems

- Exercise:
  - F → F [+ F] F [- F] F



Bracketed L-Systems

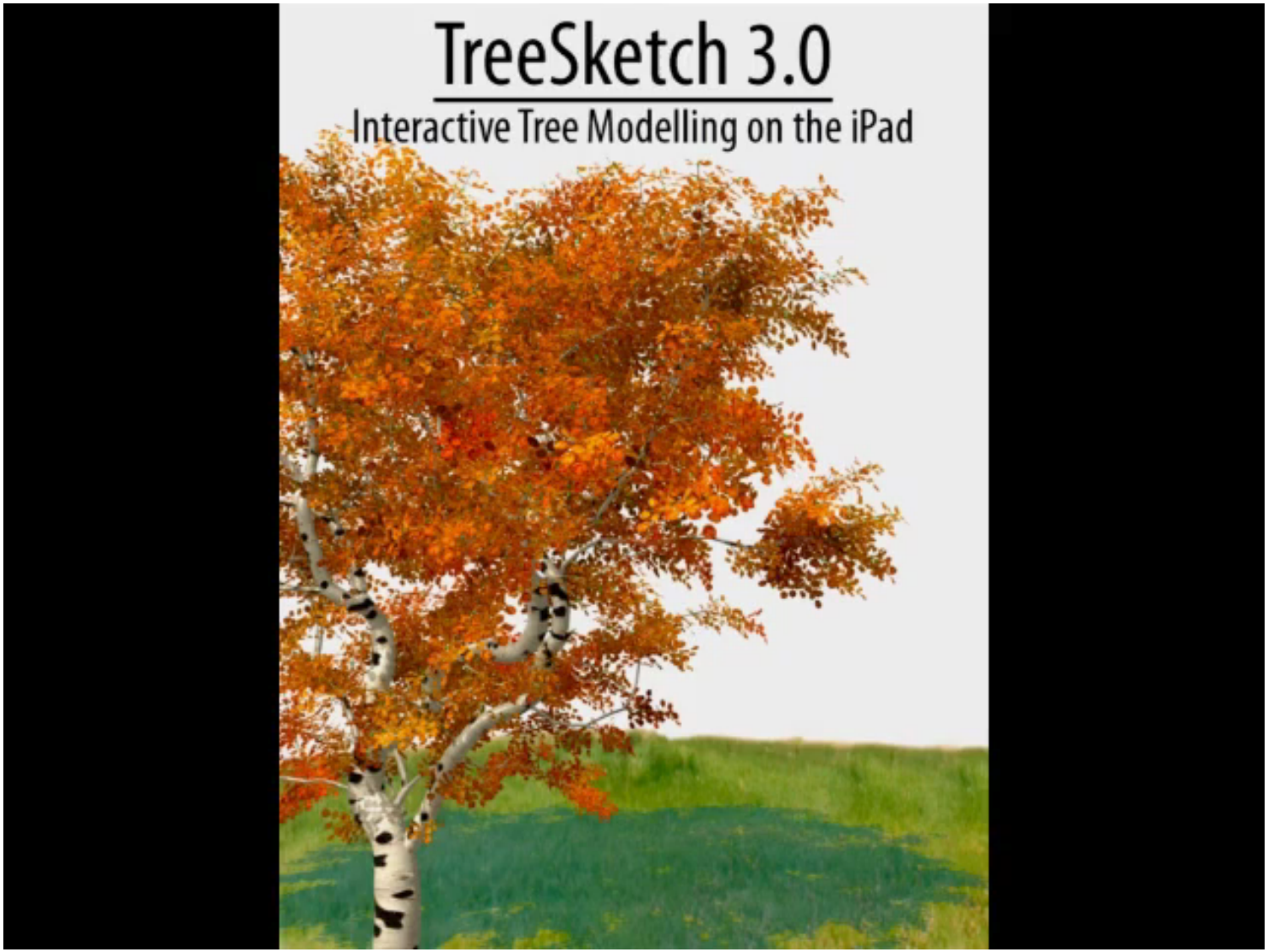
- (a) segment length remains constant
- (b) segment length decreases by 1/3 in each iteration



Applications: Plant Modeling

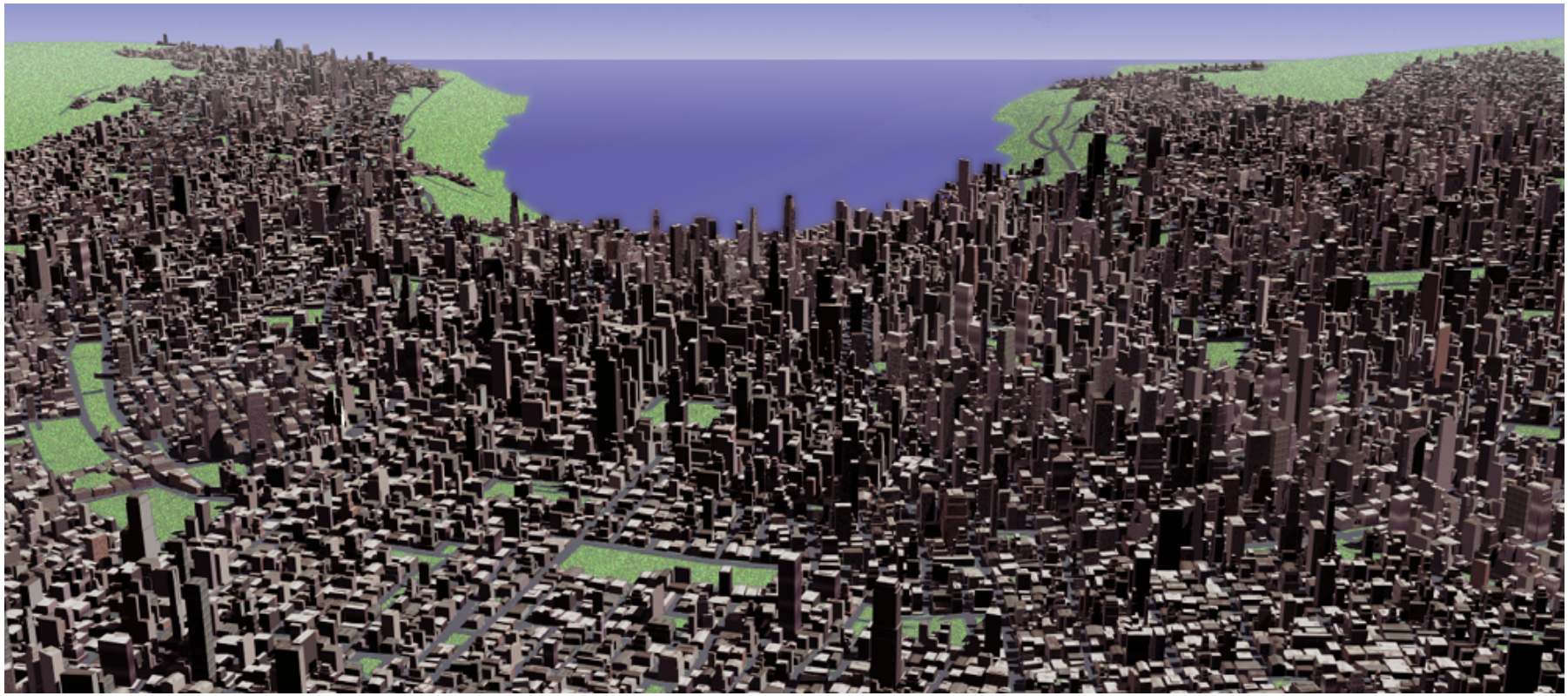
- Algorithmic Botany @ the University of Calgary
  - Covers many variants of L-Systems, formal derivations, and exhaustive coverage of different plant types.
  - http://algorithmicbotany.org/papers
  - http://algorithmicbotany.org/virtual_laboratory/

TreeSketch: Interactive Tree Modeling



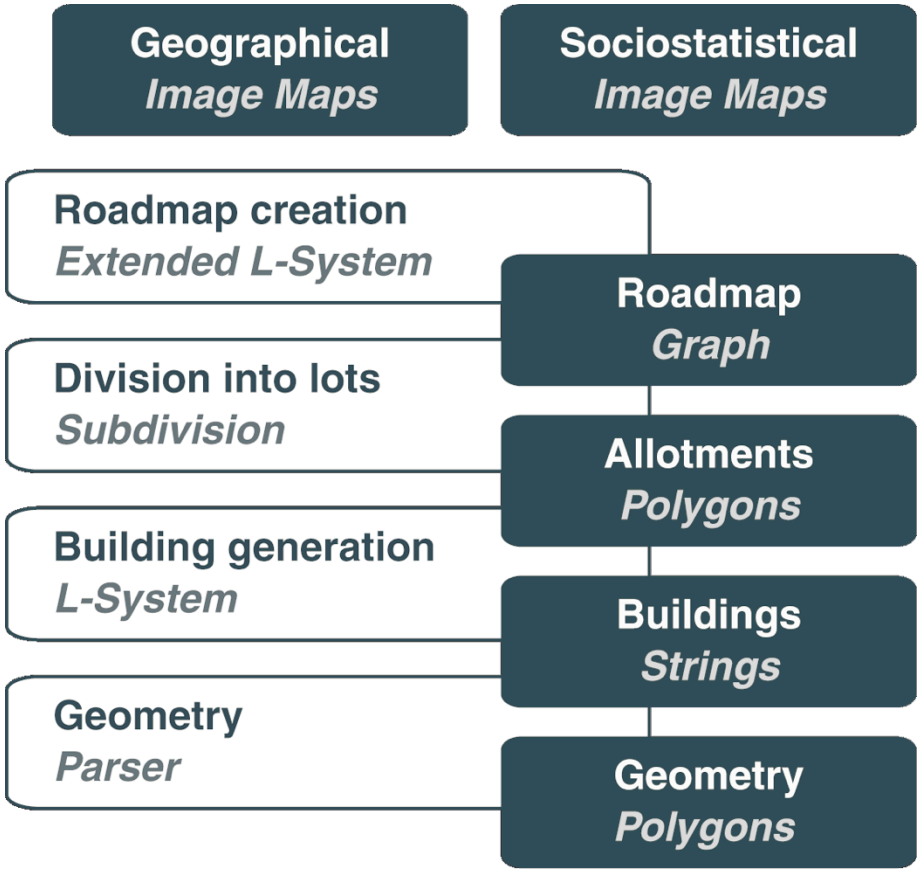
http://vimeo.com/68195050

Procedural Modeling of Cities



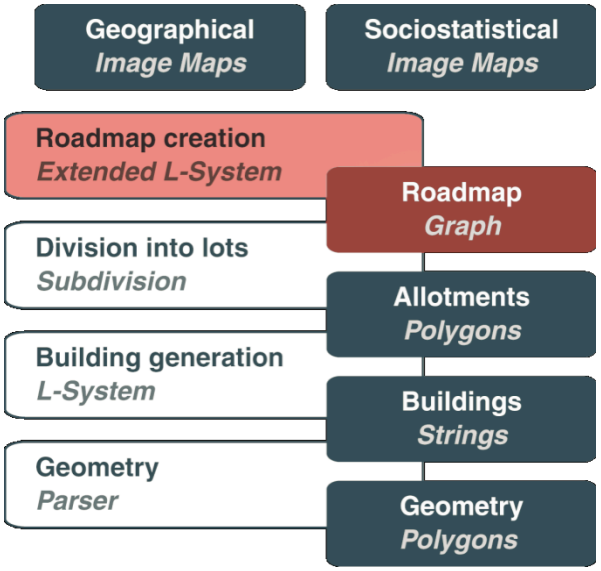
Procedural Modeling of Cities / Yoav Parish, Pascal Müller, Siggraph 2001

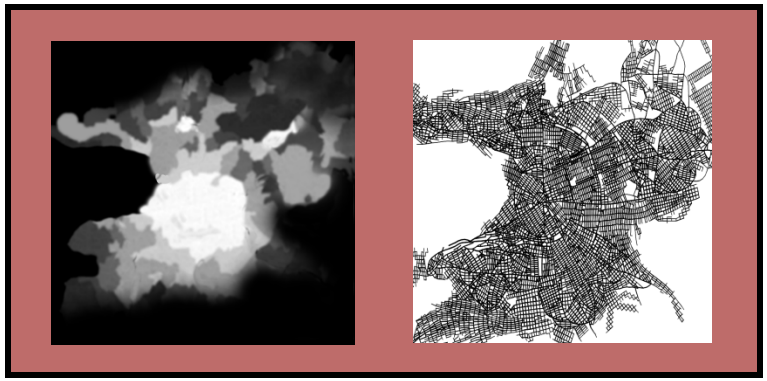
System Pipeline



Procedural Modeling of Cities / Yoav Parish, Pascal Müller, Siggraph 2001

Module 1: Streetmap Creation





- Input:
  Image maps,
  parameters for rules
- Output:
  A street graph for
  interactive editing

Procedural Modeling of Cities / Yoav Parish, Pascal Müller, Siggraph 2001

Module 1: Streetmap Creation

Figure 2: Left column: Water, elevation and population density maps of an imaginary virtual city of 20 miles diameter. Right: One possible roadmap generated from this input data.

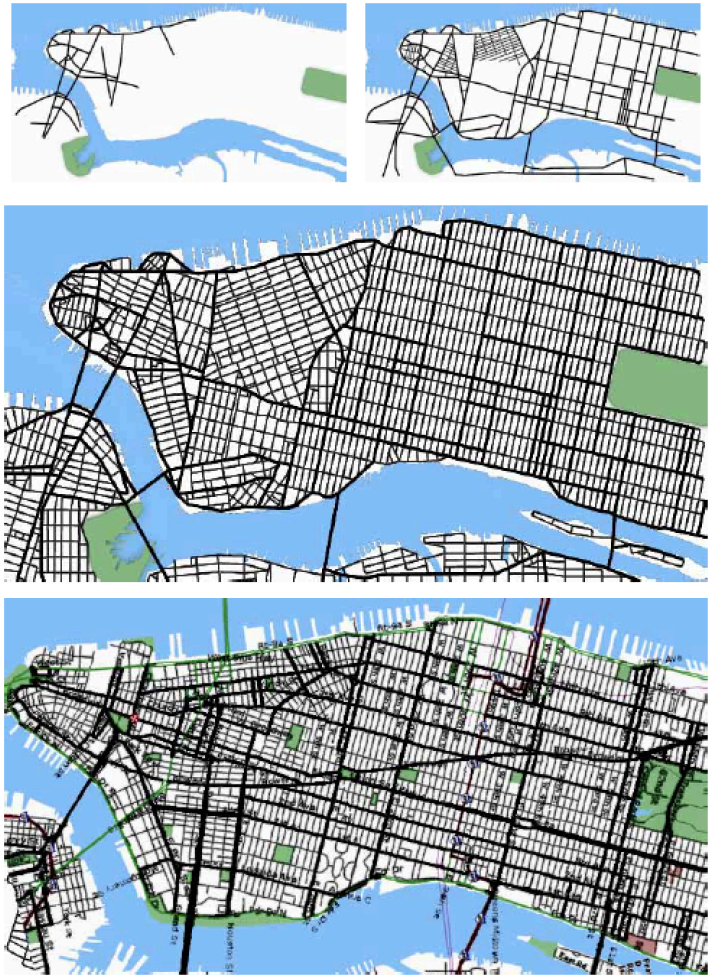
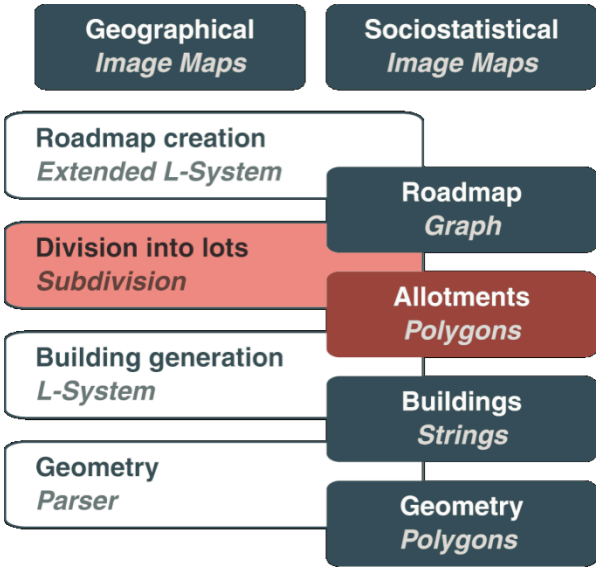
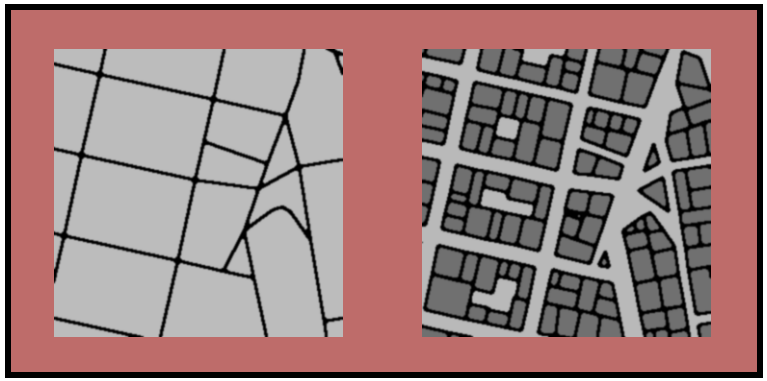


Figure 9: Street creation system applied to Manhattan. Top row: The network after 28 and 142 steps. Middle: The final roadmap. Lower: A real map of Manhattans streets for comparison.

Module 2: Division into Lots





- Input:
  Street graph, area usage map
- Output:
  Polygon set of allotments for buildings

Procedural Modeling of Cities / Yoav Parish, Pascal Müller, Siggraph 2001

Module 2: Division into Lots

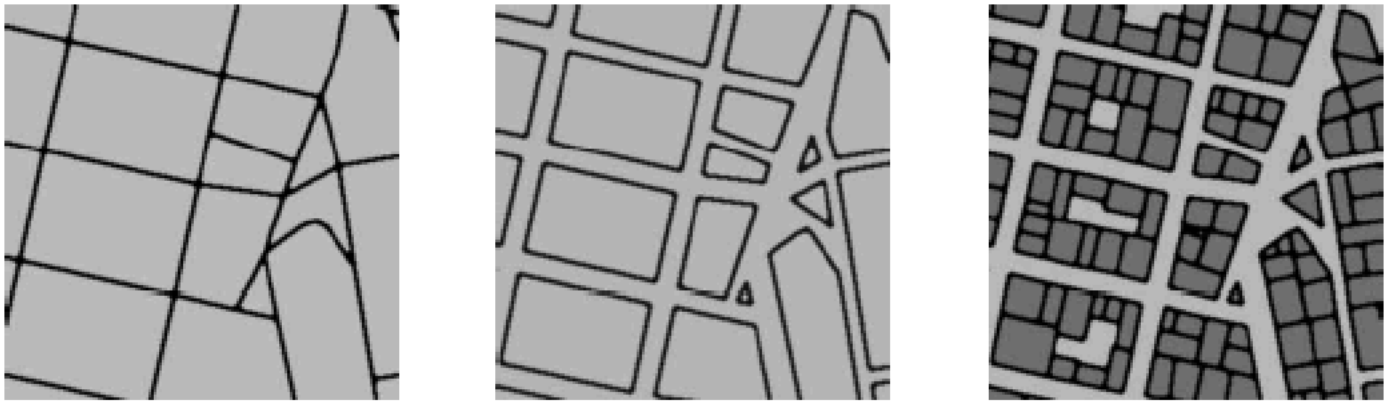
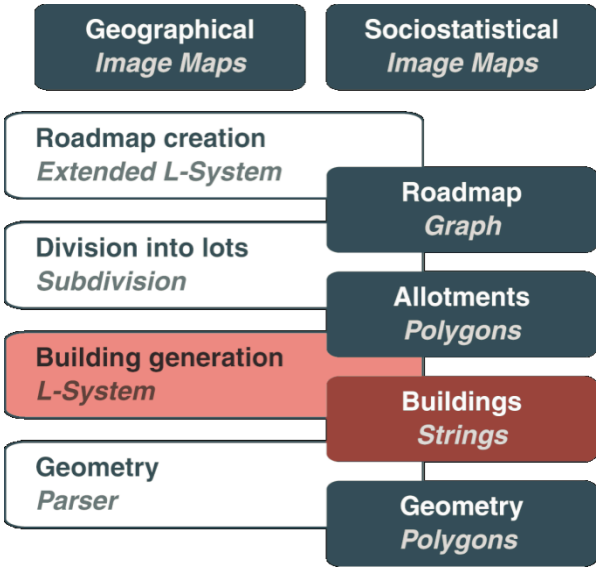
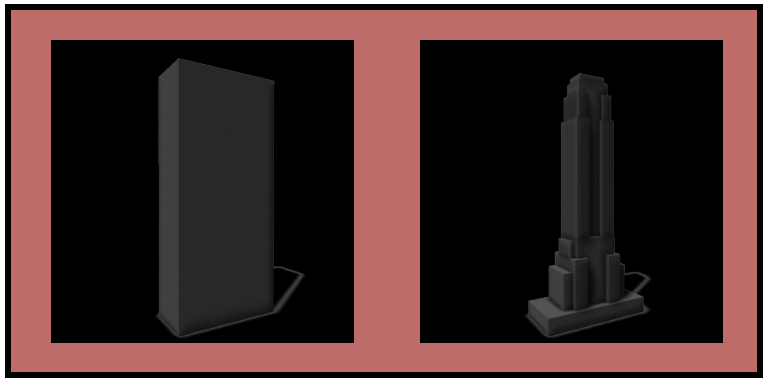


Figure 10: Left: Resulting street map. Middle: Blocks created by scaling from street crossings. Right: The generated lots. Allotments too small or without street access are removed.

Procedural Modeling of Cities / Yoav Parish, Pascal Müller, Siggraph 2001

Module 3: Building Generation





- Input:
  Lot polygons, age map and zone plan
- Output:
  Building strings with additional info

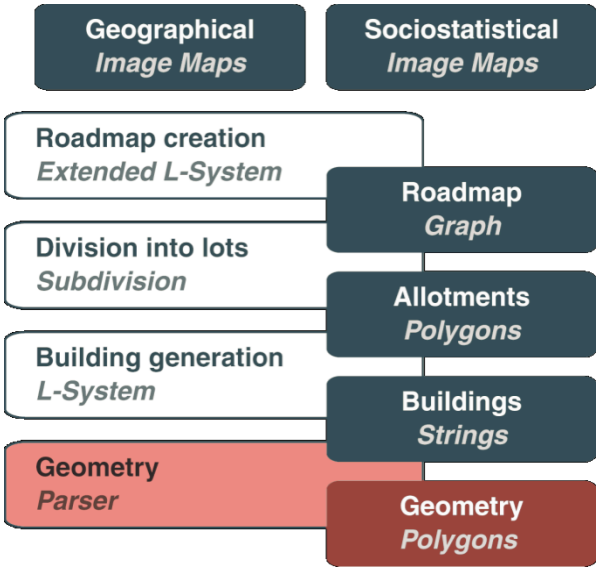
Procedural Modeling of Cities / Yoav Parish, Pascal Müller, Siggraph 2001

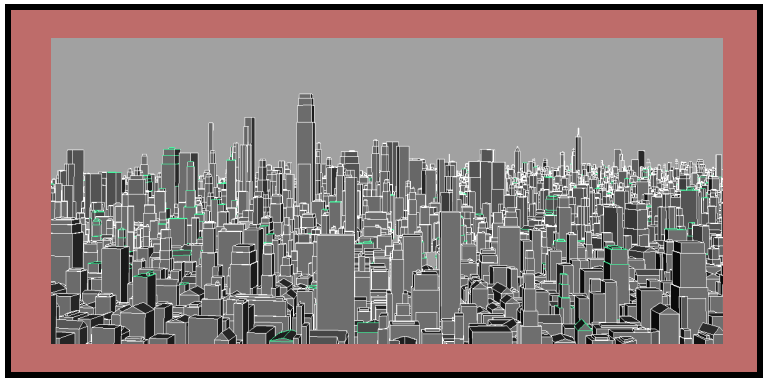
Module 3: Building Generation

Figure 11: Five consecutive steps of the generation of a building. The axiom of the L-system is the bounding box of the building, allowing easy LOD-generation.

Procedural Modeling of Cities / Yoav Parish, Pascal Müller, Siggraph 2001

Module 4: Geometry and Facades





- Input:
  Strings and building type
- Output:
  City geometry and facade texture (procedural shader)

Procedural Modeling of Cities / Yoav Parish, Pascal Müller, Siggraph 2001

Module 4: Geometry and Facades

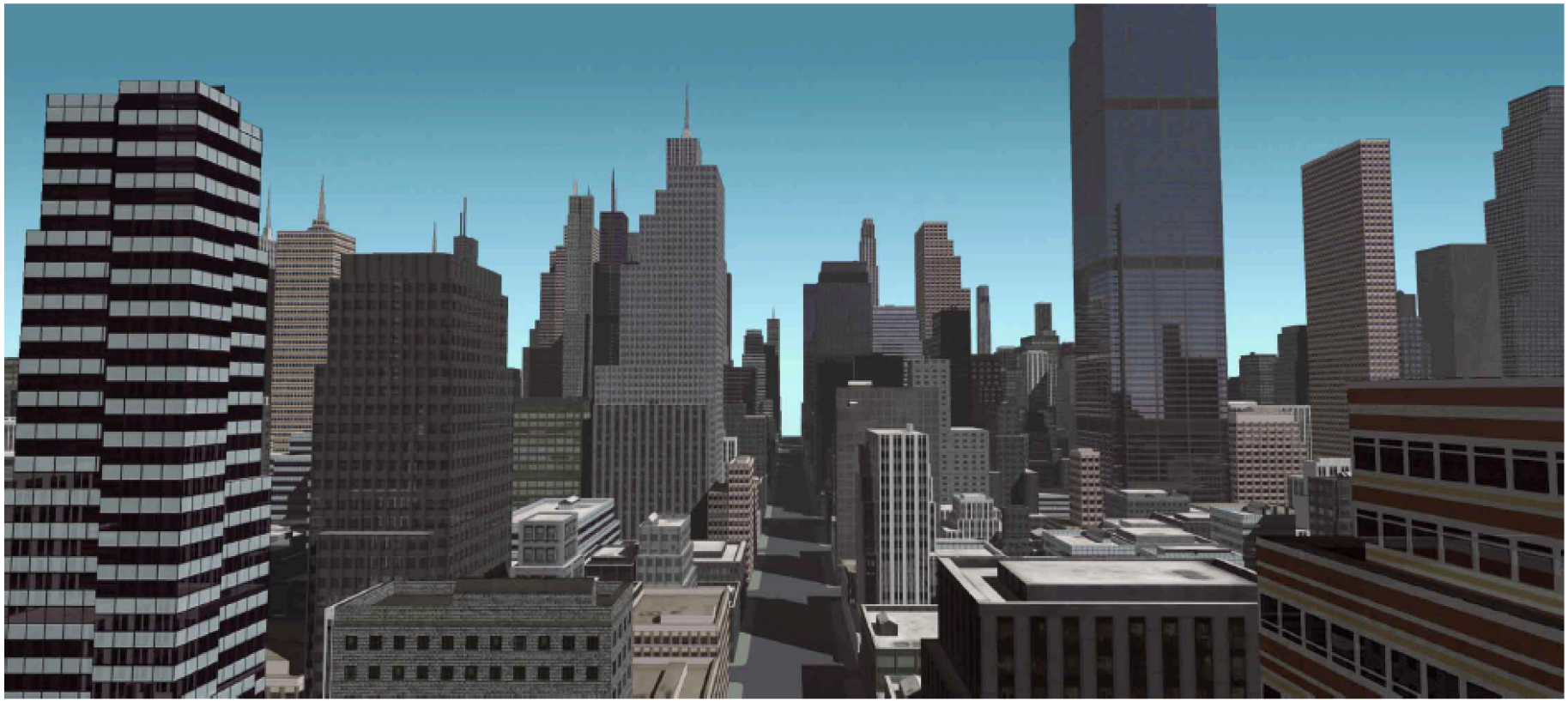


Figure 18. Somewhere in a virtual Manhattan.

Procedural Modeling of Cities / Yoav Parish, Pascal Müller, Siggraph 2001

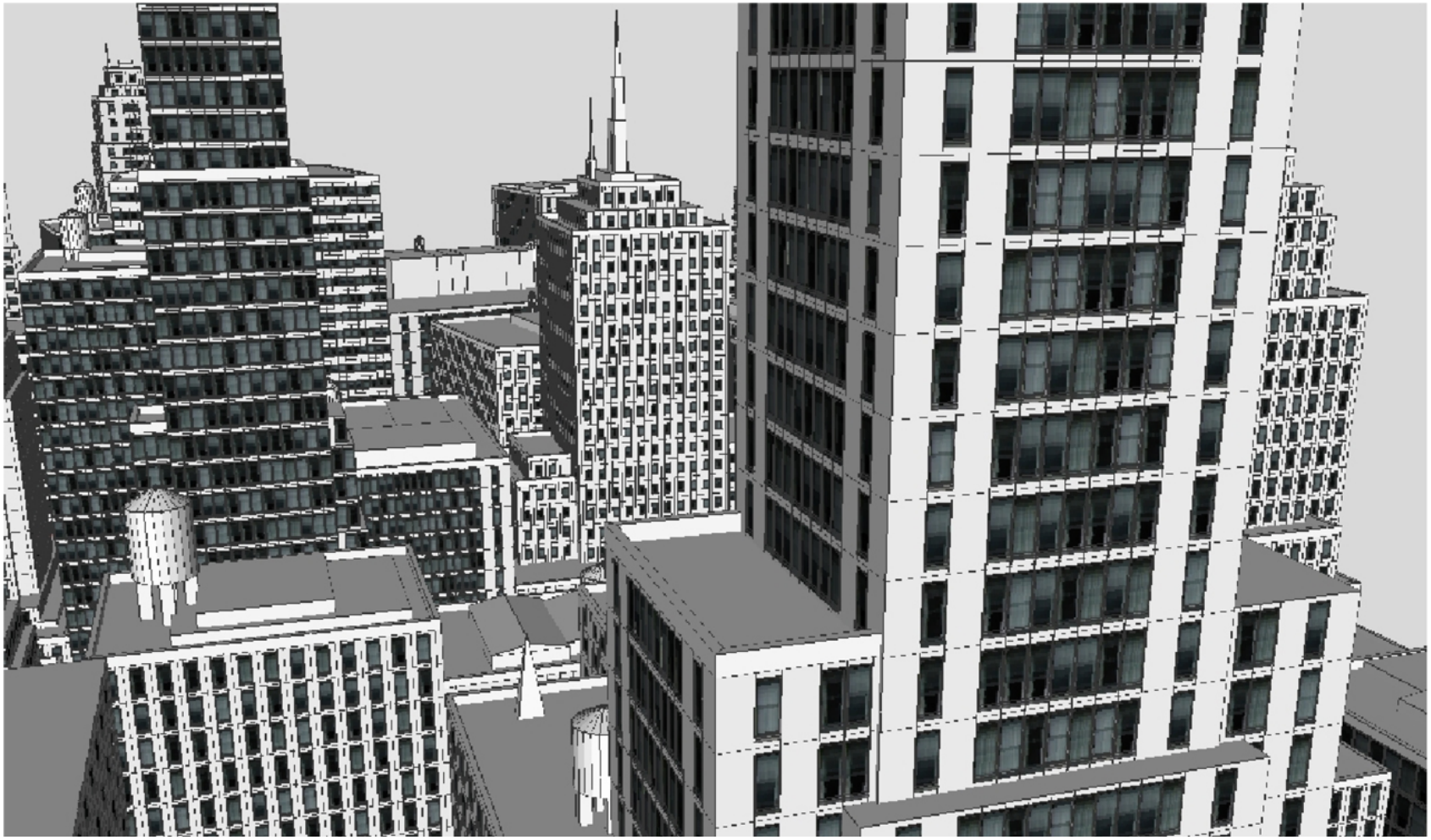
Procedural Modeling of Buildings

- Pompeii

Procedural Modeling of Buildings / Müller et al, Siggraph 2006

Procedural Modeling of Buildings

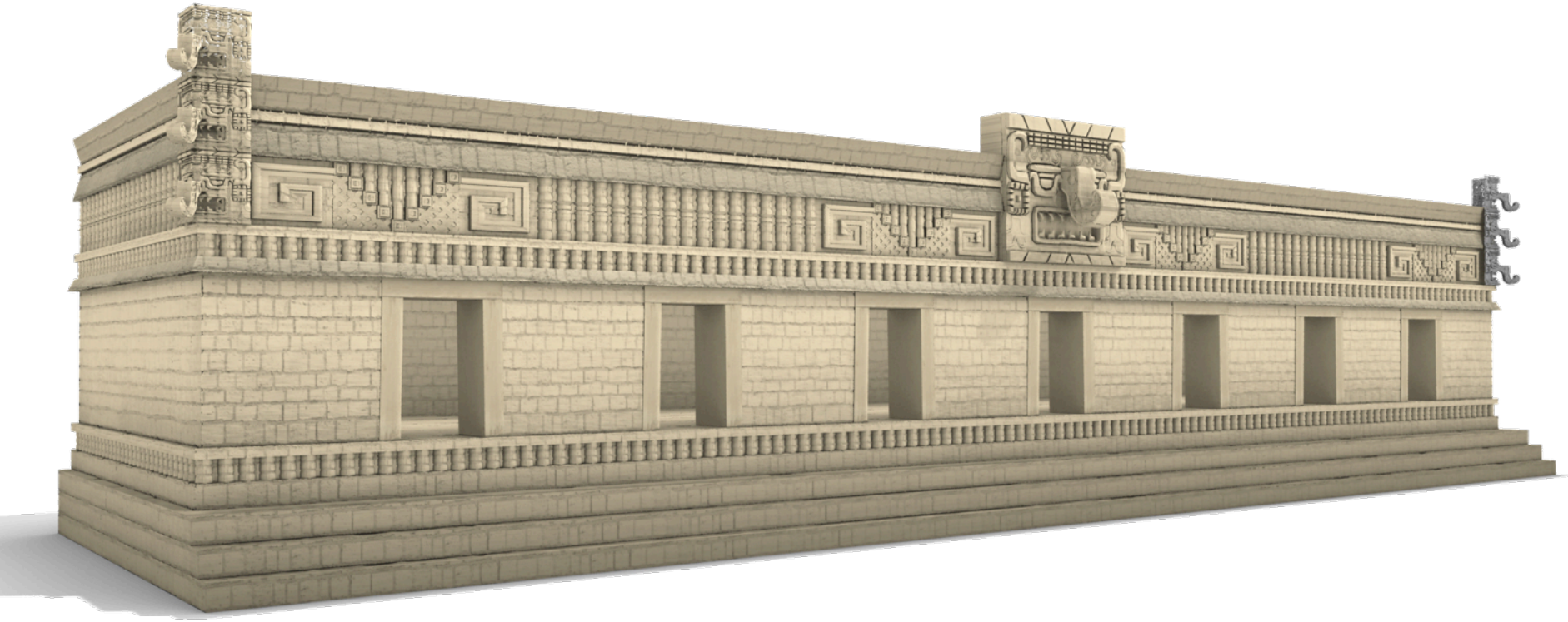
- Modern architecture



Procedural Modeling of Buildings / Müller et al, Siggraph 2006

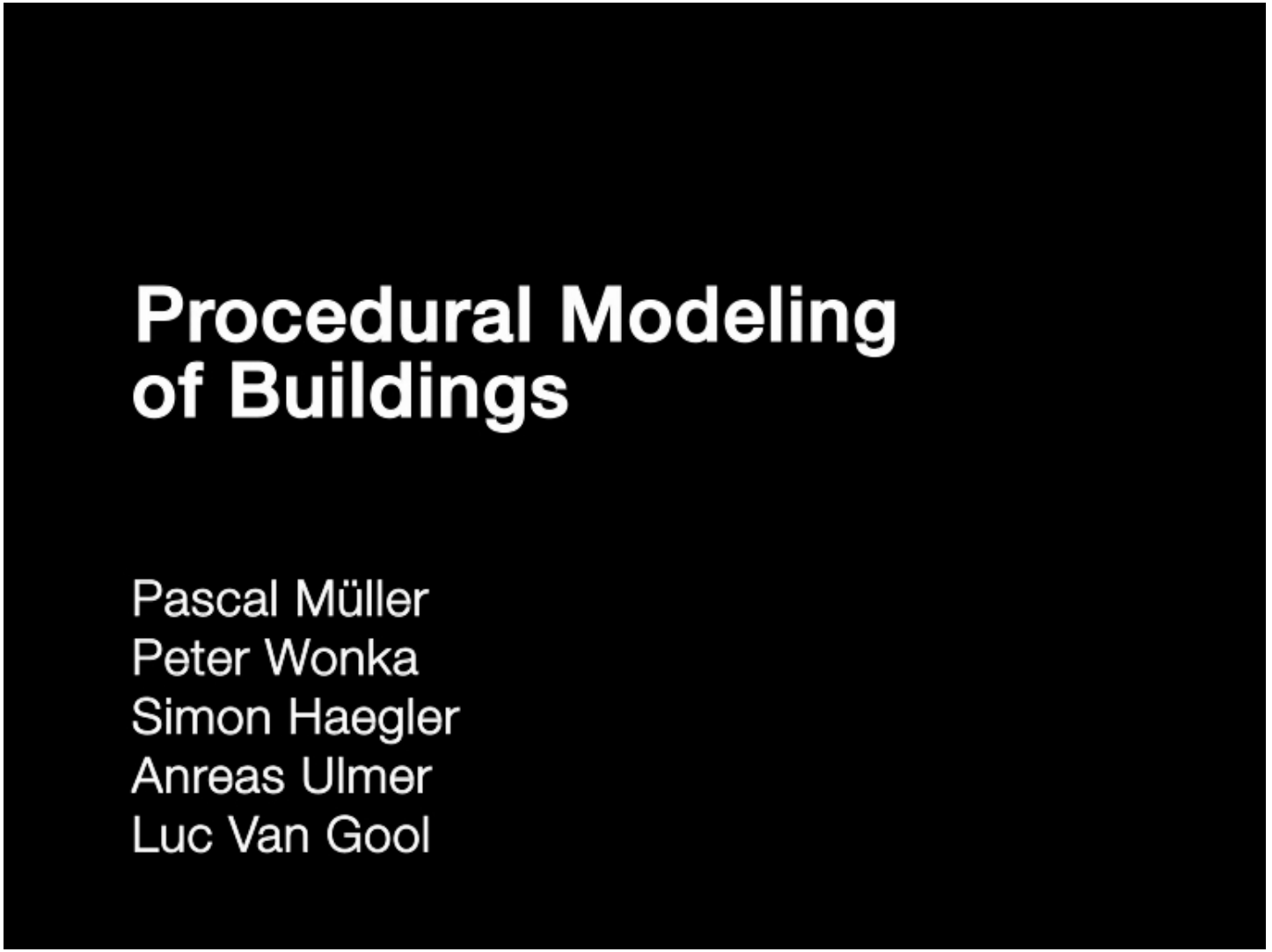
Procedural Modeling of Buildings

- Mayan architecture



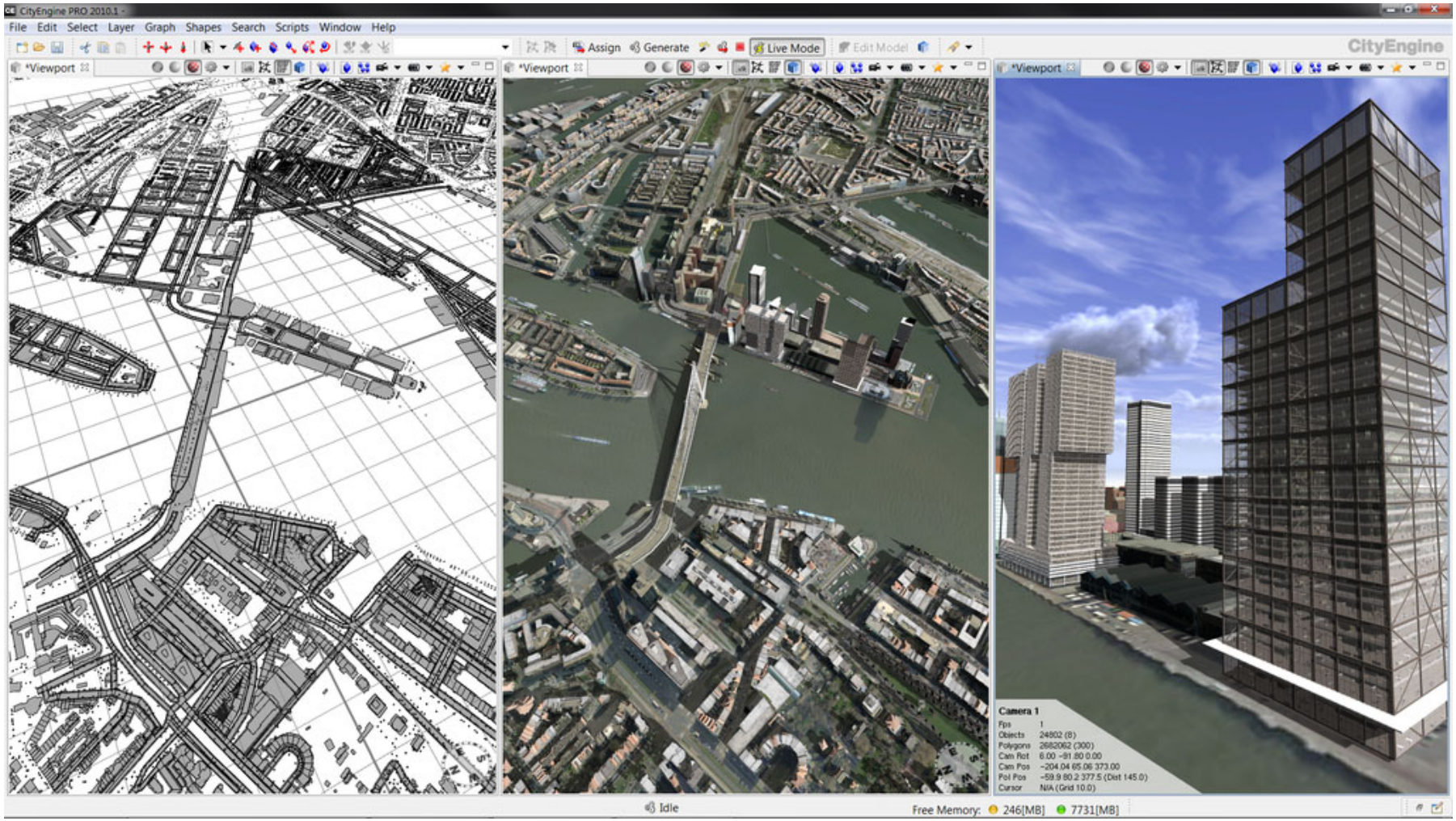
Procedural Modeling of Buildings / Müller et al, Siggraph 2006

Procedural Modeling of Buildings



Procedural Modeling of Buildings / Müller et al, Siggraph 2006

CityEngine



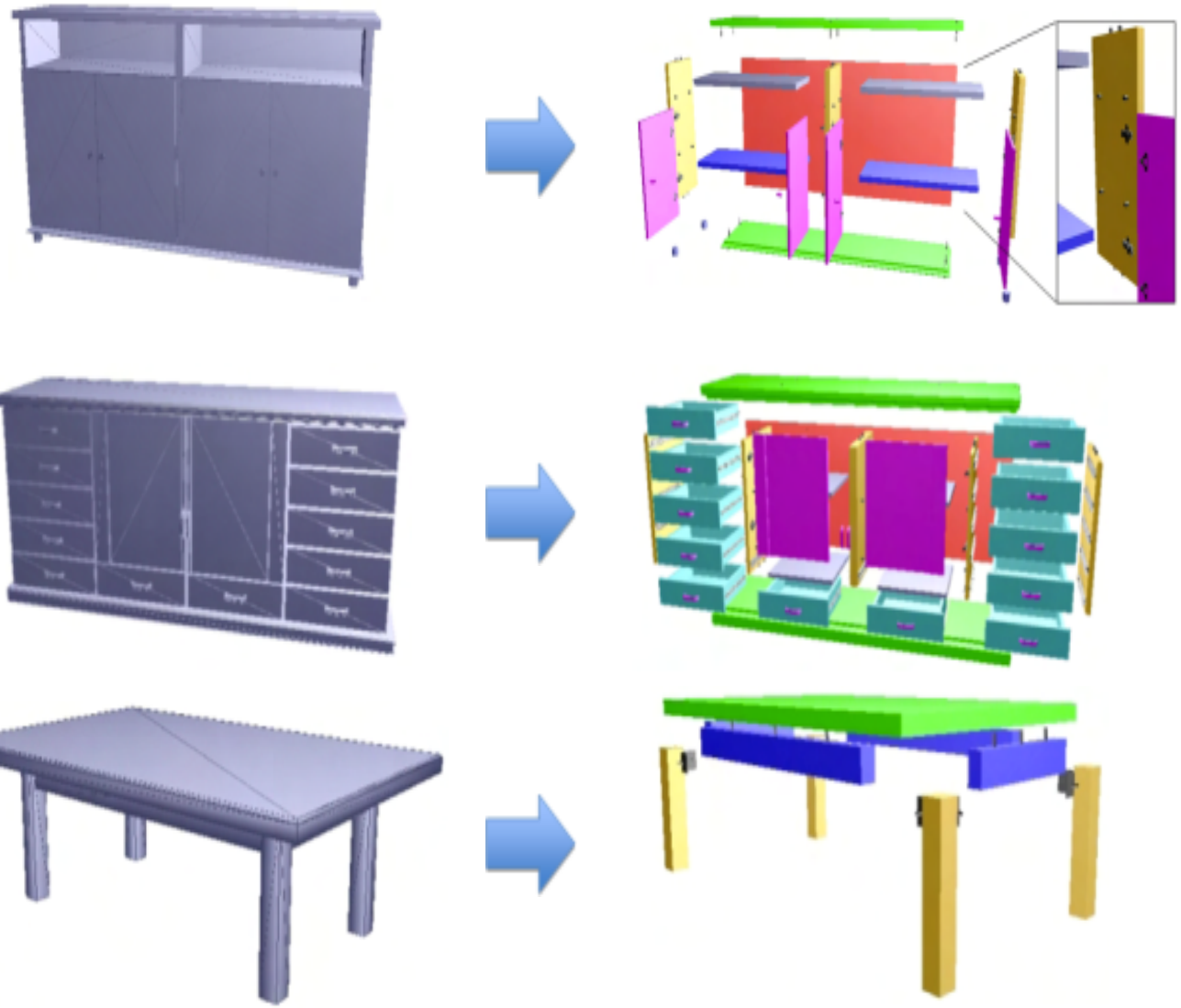
http://www.esri.com/software/cityengine/

CityEngine

http://www.youtube.com/watch?v=aFRqSJFp-I0

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

Furniture Design

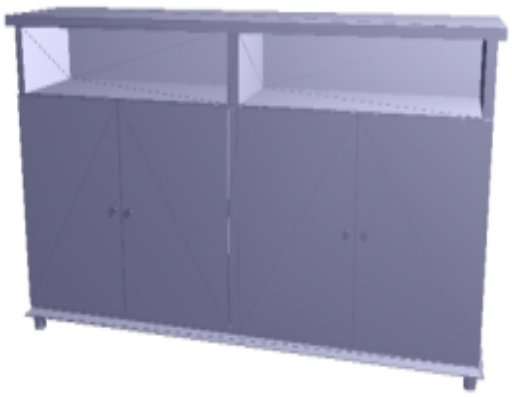


Input: 3D model

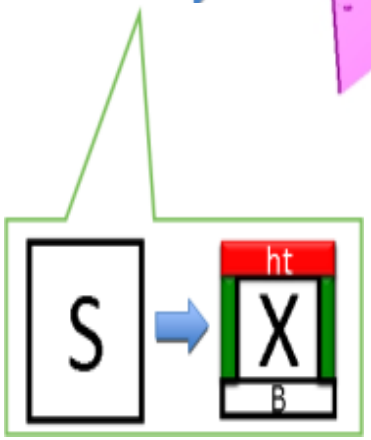
Output: Fabricatable Parts and Connectors

Converting 3D Furniture Models to Fabricable Parts and Connectors, Lau et al., Siggraph 2011

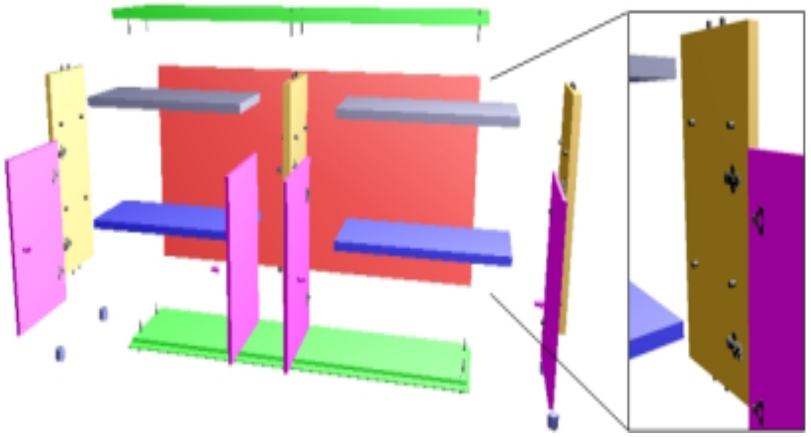
Approach



3D
model



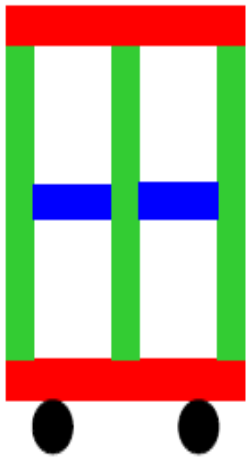
Formal
grammar



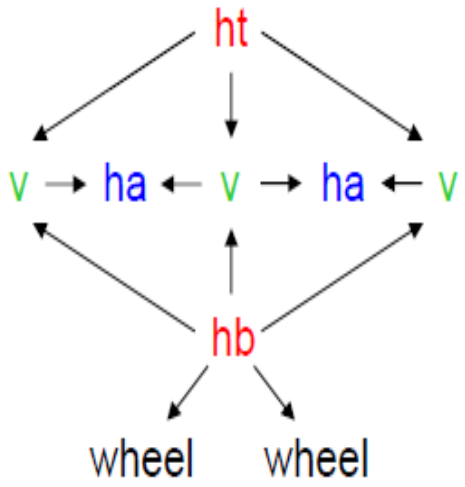
Separate parts
and
connectors

Pre-defined formal grammar to analyze structure

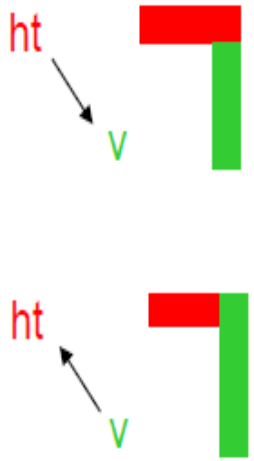
Representation of 2D Cabinets



Example 2D Cabinet



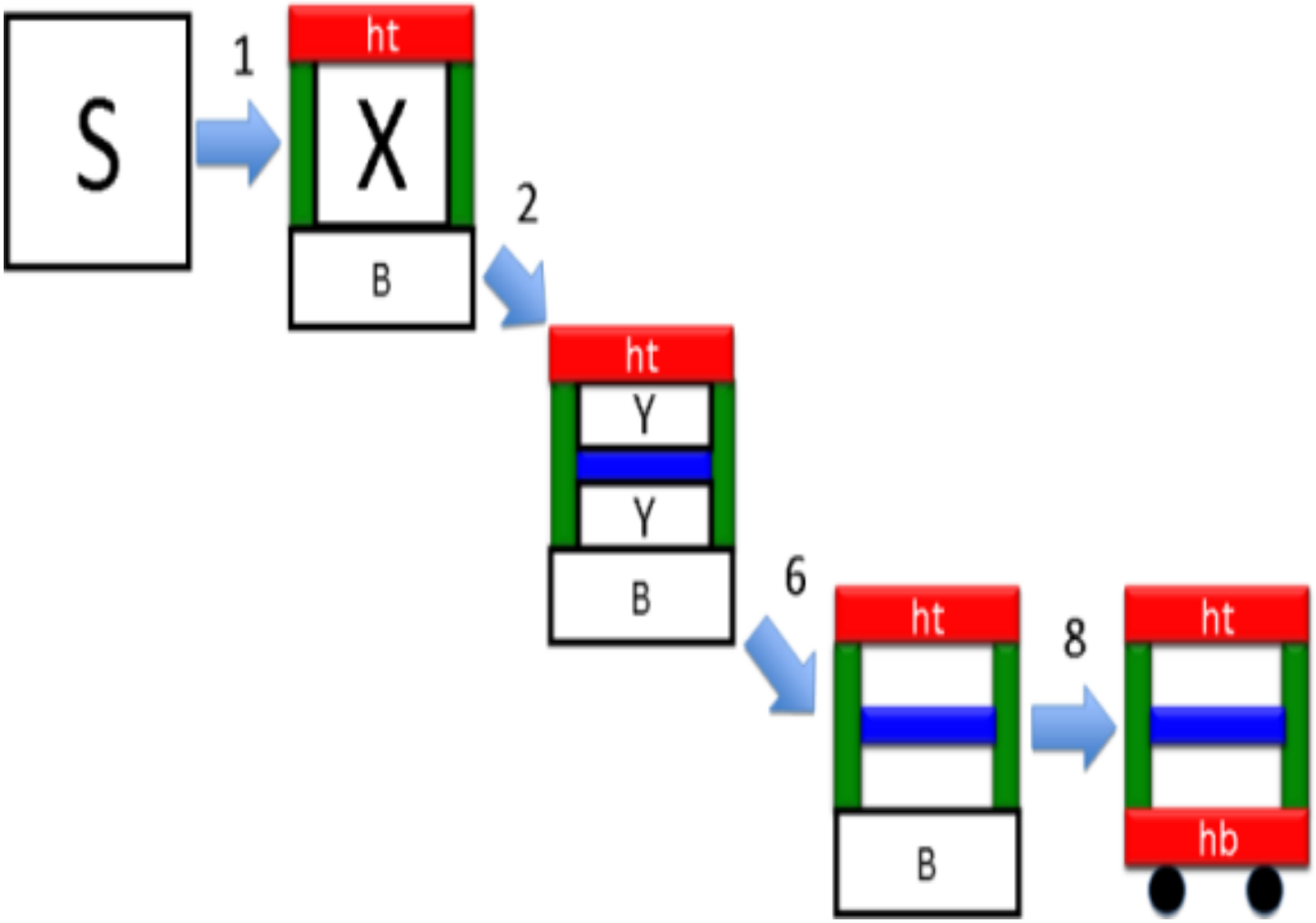
Corresponding Graph



Positioning of Parts

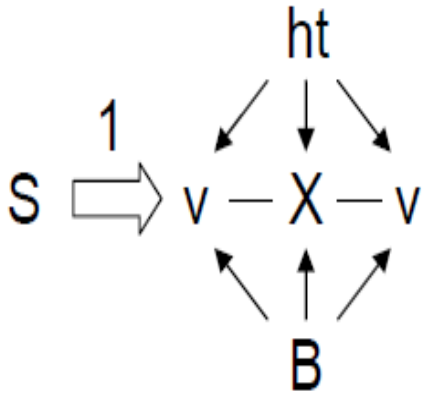
ht: horizontal-top, hb: horizontal-bottom hb: horizontal-bottom, v: vertical

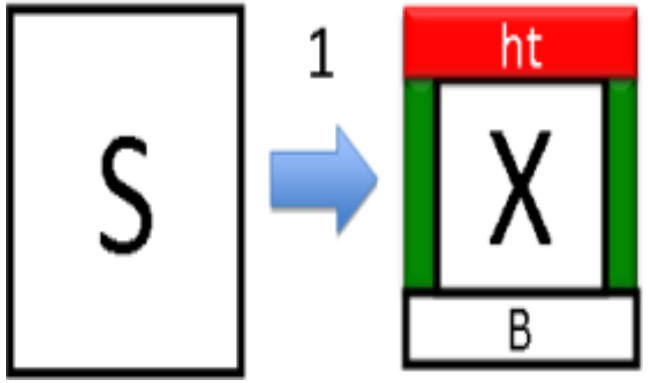
Sequence of Production Rules



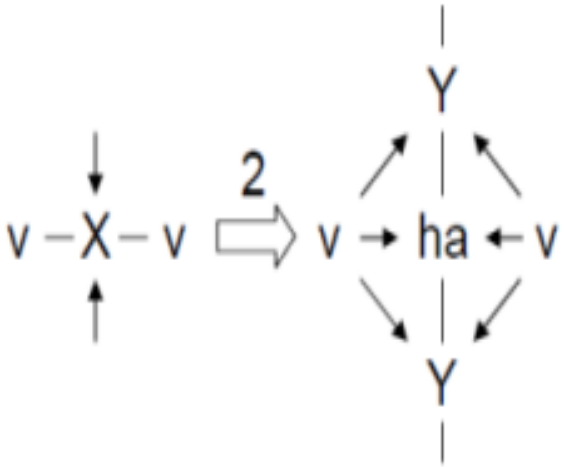
Examples of Production Rules

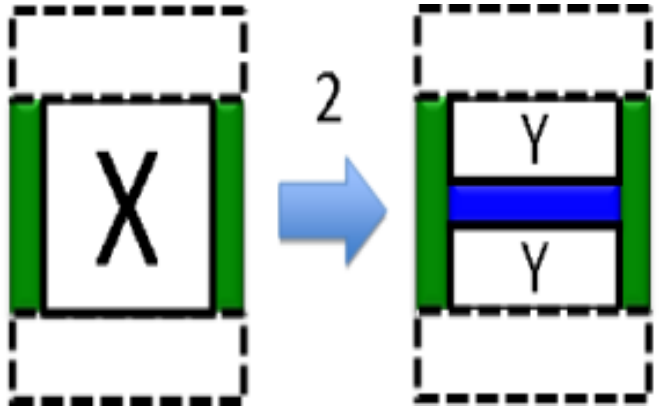
Production Rule 1





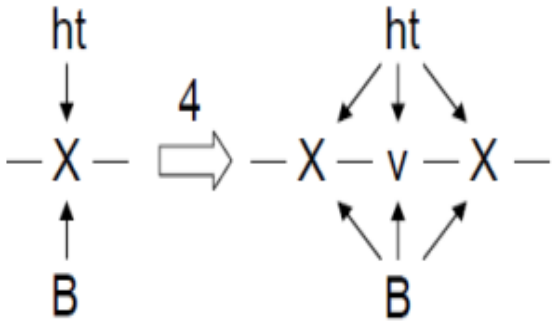
Production Rule 2

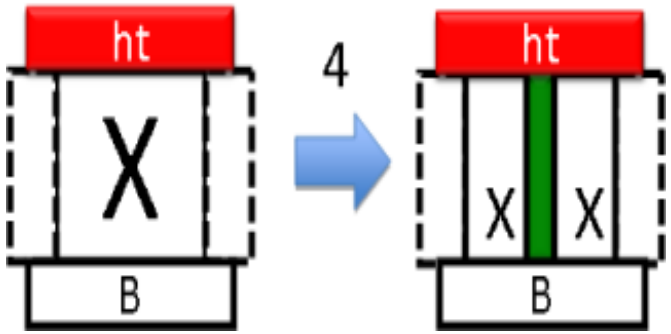




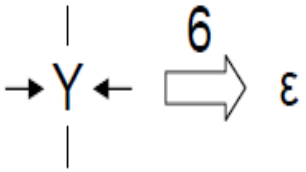
Examples of Production Rules

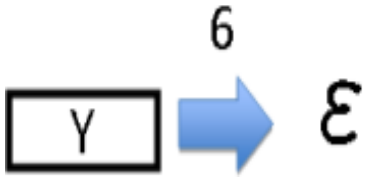
Production Rule 4



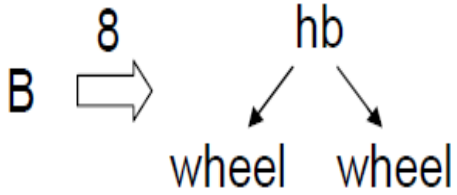


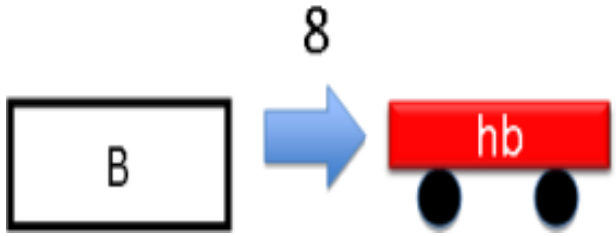
Production Rule 6



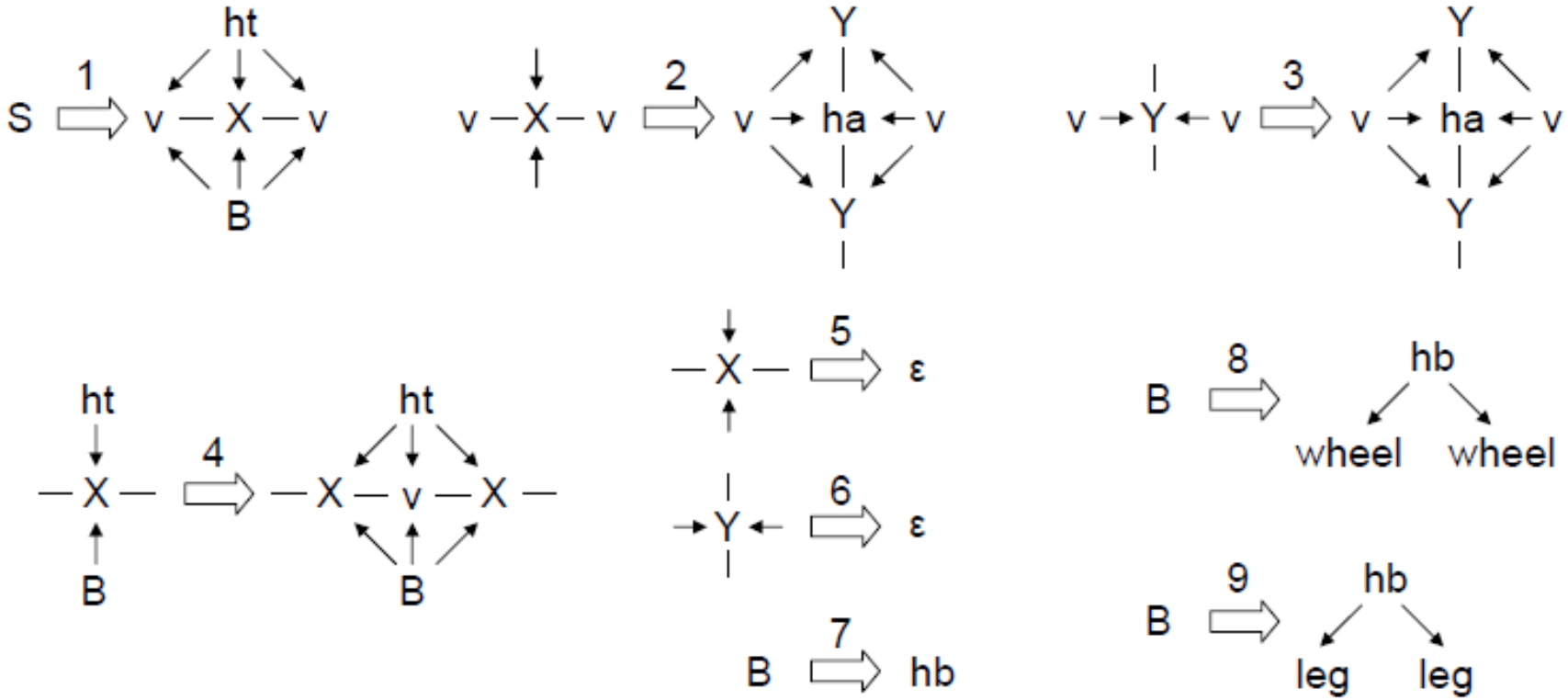


Production Rule 8





All Production Rules



Formal Grammar for 2D Cabinets

$N = \{S, B, X, Y\}$

Non-terminal Symbols
- Collection of Parts, X: compartment separated by vertical wall. Y: compartment separated by horizontal wall.

$\sum = \{hb, ht, v, ha, leg, wheel\}$

Terminal Symbols
- Separate Parts

P : Set of Production Rules

S : Start Symbol

The language specifies a directed graph,
and each graph represents parts and connectors

Lexical Analysis

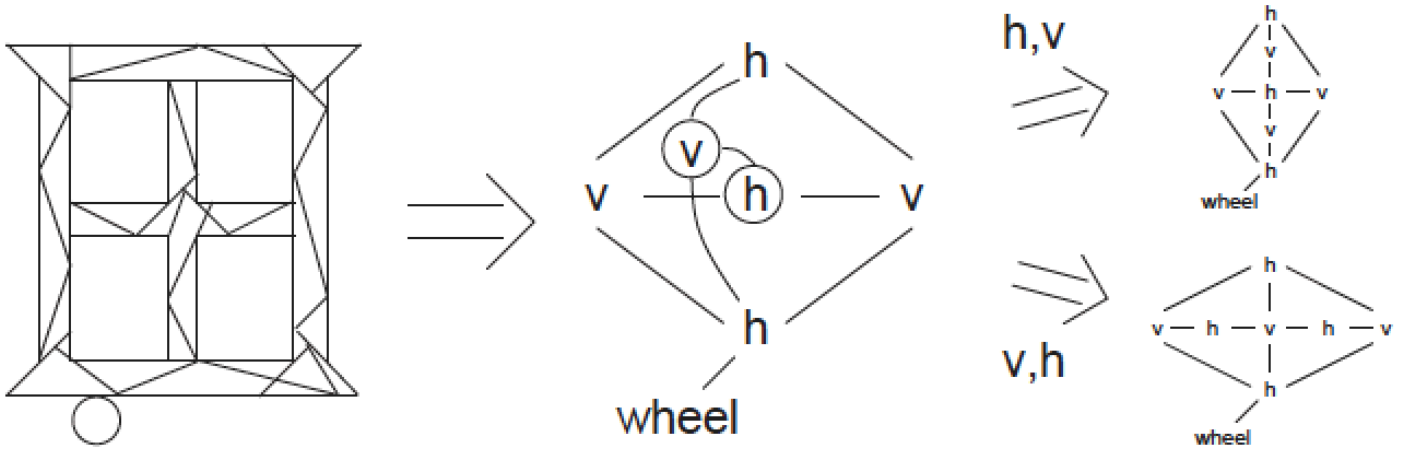


Figure 4: *Lexical Analysis: The input can be an irregularly-shaped mesh (left). We convert the mesh to a voxel representation and fit primitive shapes to it. The resulting shape of each part may not necessarily have that exact primitive shape (i.e. top and bottom horizontal pieces). The circled parts v and h (middle) completely cross through each other, and we parse them again in either order to create two possible primitive graphs (right).*

Results: IKEA ALVE Cabinet

Questions?

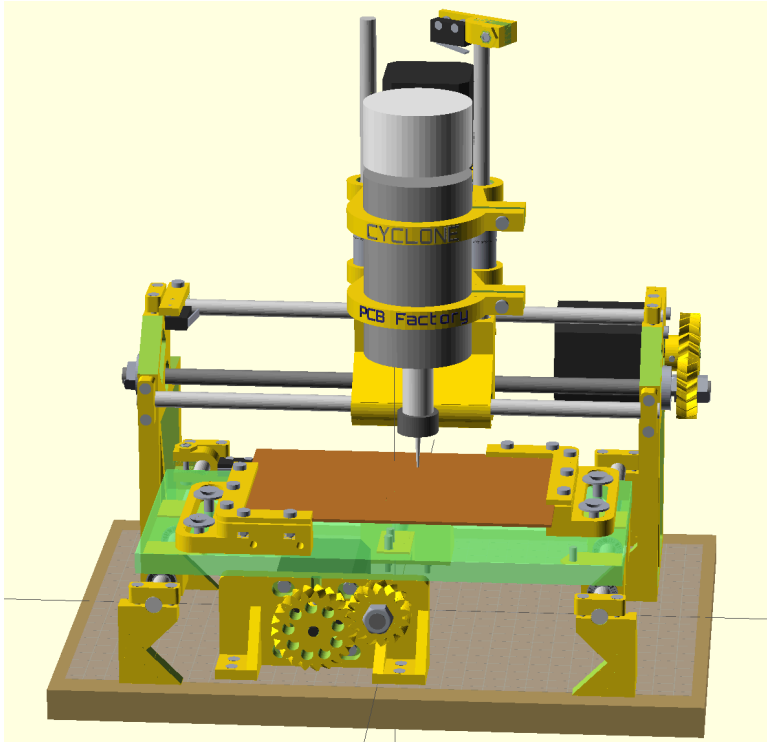
- Any disadvantage of procedural modeling?

The Plan For Today

- Constructive Solid Geometry (CSG)
- Procedural Modeling
- OpenSCAD

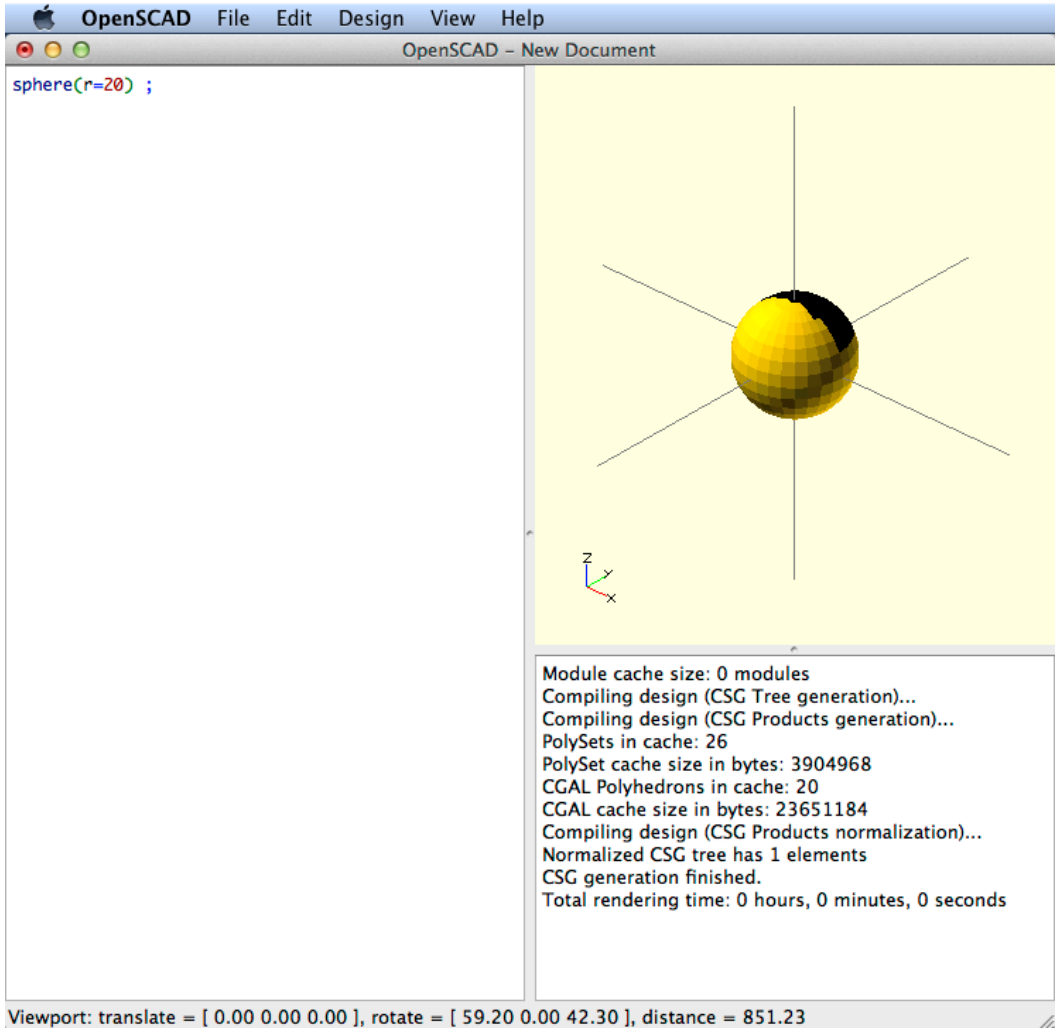
OpenSCAD

- 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



OpenSCAD

- Interface
  - 3 panels
    - Script
    - View
    - Info
- Compile (F5)
  - Design->Compile
- Show Axes (Ctrl+2)



OpenSCAD CheatSheet

OpenSCAD CheatSheet

Syntax

```
var = value;
module name(…) { … }
name();
function name(…) = …
name();
include <….scad>
use <….scad>
```

2D

```
circle(radius)
square(size,center)
square([width,height],center)
polygon([points])
polygon([points],[paths])
```

3D

```
sphere(radius)
cube(size)
cube([width,height,depth])
cylinder(h,r,center)
cylinder(h,r1,r2,center)
polyhedron(points, triangles, convexity)
```

Transformations

```
translate([x,y,z])
rotate([x,y,z])
scale([x,y,z])
resize([x,y,z],auto)
mirror([x,y,z])
multmatrix(m)
color("colorname")
color([r, g, b, a])
hull()
minkowski()
```

Boolean operations

```
union()
difference()
intersection()
```

Modifier Characters

```
*   disable
!   show only
#   highlight
%   transparent
```

Mathematical

```
abs
sign
acos
asin
atan
atan2
sin
cos
floor
round
ceil
ln
len
log
lookup
min
max
pow
sqrt
exp
rands
```

Other

```
echo(…)
str(…)
for (i = [start:end]) { … }
for (i = [start:step:end]) { … }
for (i = […,…,…]) { … }
intersection_for(i = [start:end]) { … }
intersection_for(i = [start:step:end]) { … }
intersection_for(i = […,…,…]) { … }
if (…) { … }
assign (…) { … }
search(…)
import("….stl")
linear_extrude(height,center,convexity,twist,slices)
rotate_extrude(convexity)
surface(file = "….dat",center,convexity)
projection(cut)
render(convexity)
```

Special variables

```
$fa minimum angle
$fs minimum size
$fn number of fragments
$t  animation step
```

Links

- Official website
- Manual
- MCAD library
- Other links

Examples

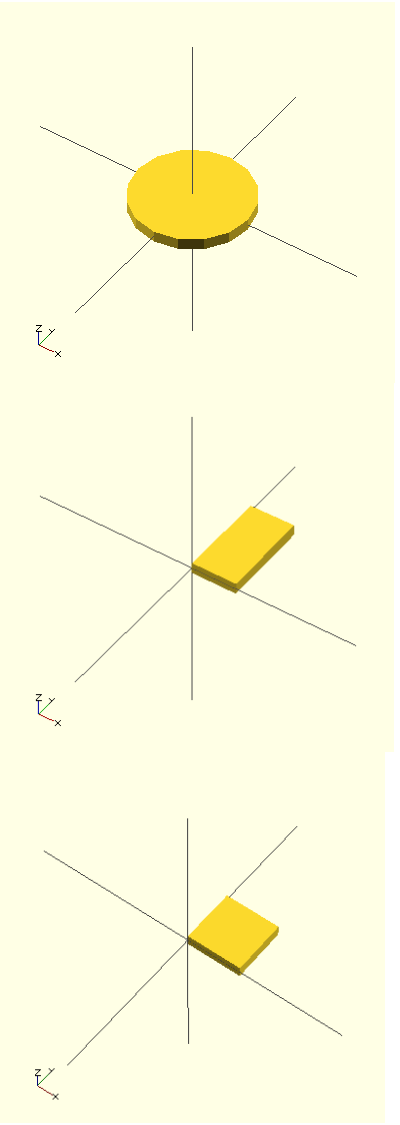
```
cylinder(10,5,5);
cylinder(h=10,r=5);
```

Code / Issues

By Peter Uithoven @ Fablab Amersfoort

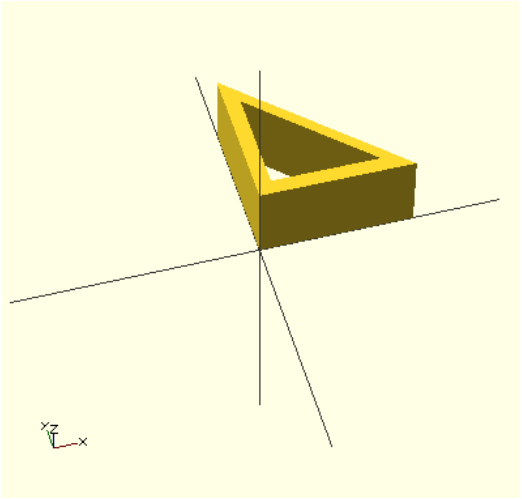
2D Primitives

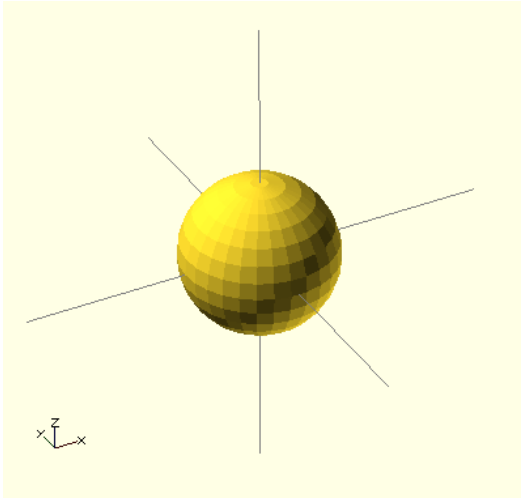
- Circle
  - `circle(5);`
  - `circle(r=5);`
- Square
  - `square(5);`
  - `square([4,8]);`
- Polygon
  - Need to specify points and paths, in this format: polygon([points],[paths]);
    - e.g., `polygon( [ [0,0],[5,0],[5,5], [0,5] ] , [ [0,1,2,3] ]);`
    - path is an optional parameter, assume in order if omitted
- Notes:
  - Remember the ";" which
  - Thickness is 1mm
  - Use "[" and "]" to pass multiple values



2D to 3D Extrusion

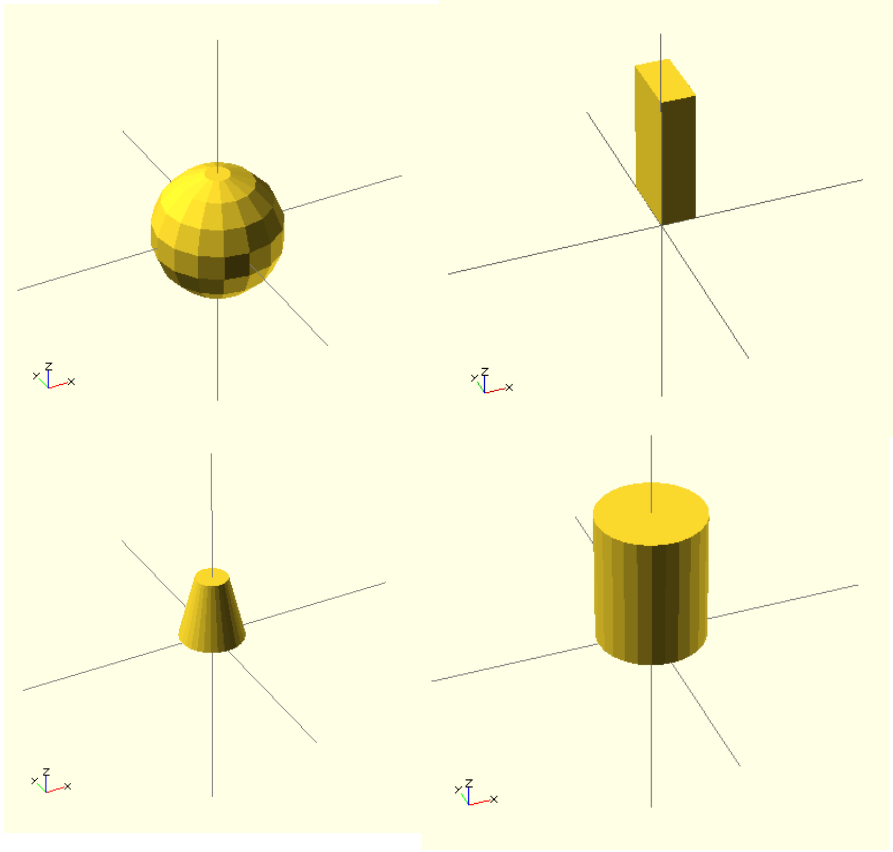
- Linear extrusion
  - `linear_extrude(50);` or
  - `linear_extrude(height=50);`
- Rotational extrusion
  - Revolves a 2D shape around the Z axis
  - `rotate_extrude() circle(20);`





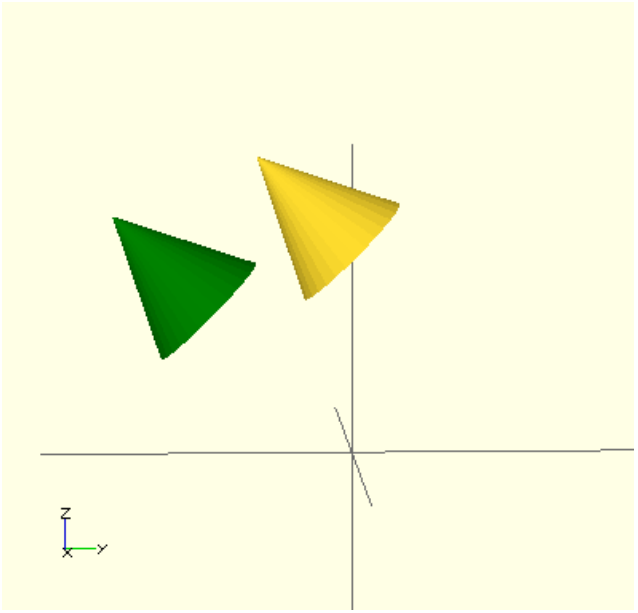
3D Primitives

- Sphere
  - `sphere(5);`
    `sphere(r=5);`
- Cube
  - `cube(5);`
  - `cube([4,8,16]);`
- Cylinder
  - `cylinder(20,10,5);`
    `cylinder(h = 20,`
    `r1 = 10, r2 = 5);`
  - `cylinder(h=20,r=10);`



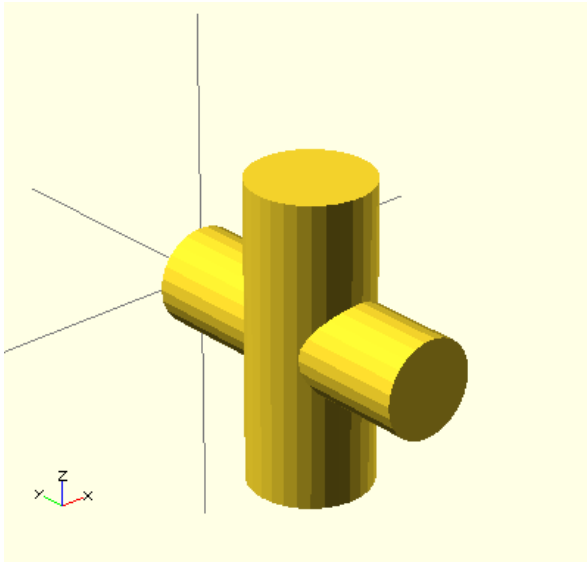
Transformations

- Translate
  - e.g., `translate([10,0,0])`
    `sphere(5); // translate`
    `along x axis`
- Rotate
- Scale
- Order dependent
  - `translate([0,0,10])`
    `rotate([45,0,0])`
    `cylinder([20,10,0]);`
  - `Color("green")`
    `rotate([45,0,0])`
    `translate([0,0,10])`
    `cylinder([20,10,0]);`



CSG

- Union
- Intersection
- Difference
- Example:

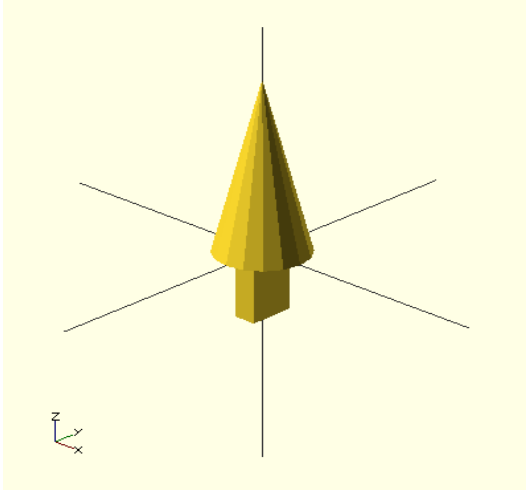


```
union()
{
        translate([0,-25,-25]) cylinder(50,10,10);
        rotate([90,0,0]) cylinder(50,8,8);
}
```

Module

- Procedures/Functions

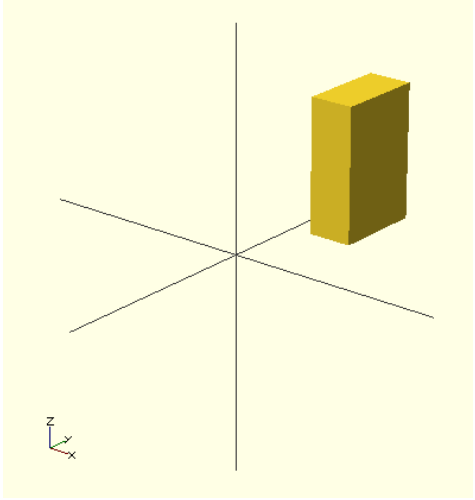
```
module leaves() { cylinder(20,5,0); }
module box() { cube([5,10,15]); }
module tree() {
      leaves();
      scale([0.5,0.5,0.5]) translate([-2.5,-5,-
      15]) box();
   }
tree();
```



Module

- Parameters

```
module box(w,l,h,tx,ty,tz){
      translate([tx,ty,tz])
cube([w,l,h]);
}
box(5,10,15,10,0,5);
```



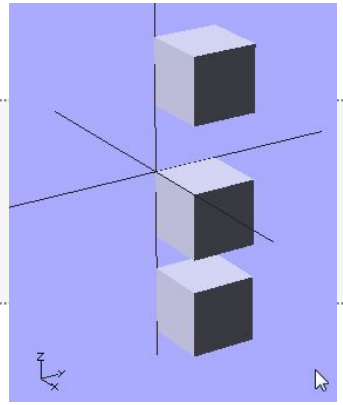
- Default values

```
module box2(w=5,l=10,h=20){
      echo("w=", w, " l=", l, " h=", h);
      cube([w,l,h]);
}
box2();
```

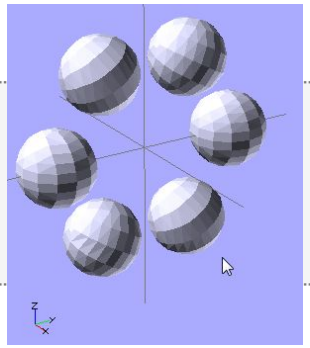
Loops

```
for (loop_variable_name = range or vector) {
        …..
      }
```

```
for ( z = [-1, 1, -2.5]) {
 translate( [0, 0, z] )
    cube(size = 1, center = false);
}
```

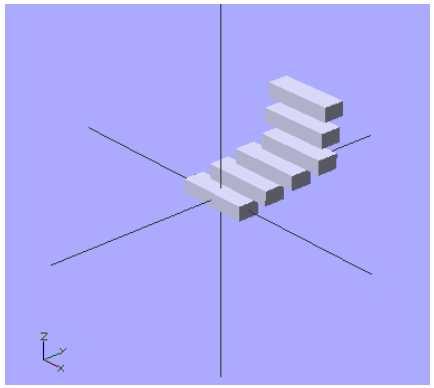


```
 for ( i = [0:5] ) {
  rotate( i*360/6, [1, 0, 0])
   translate( [0, 10, 0] ) sphere(r = 1);
 }
```

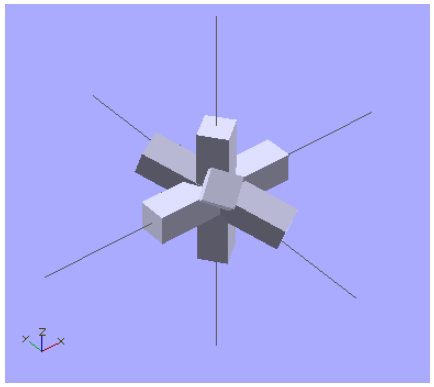


Loops

```
 for(i = [ [ 0,  0,  0],
           [10, 12, 10],
           [20, 24, 20],
           [30, 36, 30],
           [20, 48, 40],
           [10, 60, 50] ])
{
    translate(i)
    cube([50, 15, 10], center = true);
}
```



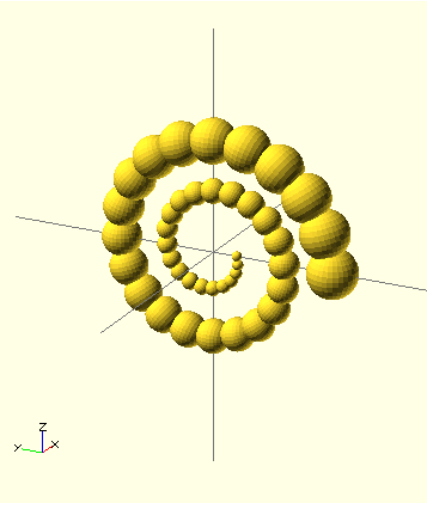
```
for(i = [ [  0,  0,   0],
          [ 10, 20, 300],
          [200, 40,  57],
          [ 20, 88,  57] ])
{
    rotate(i)
    cube([100, 20, 20], center = true);
}
```



Variables

- Assign() statement
  - In openscad, one can only assign variables at file top-level or module top-level
  - If you need it inside the for loop, you need to use assign(), e.g,:

```
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);
    }
```



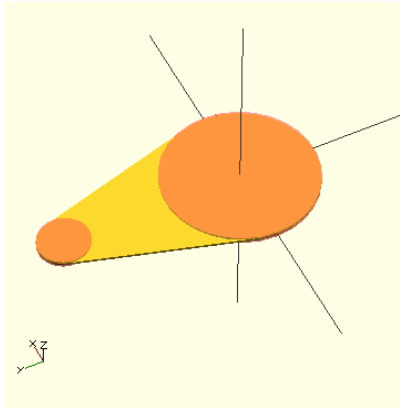
Conditionals

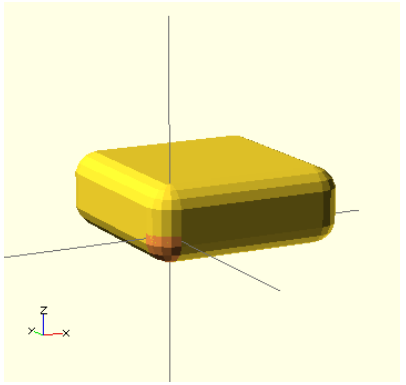
- If/else/else if
  - Syntax similar to C/C++

```
if (boolean_expression) { .... }
if (boolean_expression) { .... } else {.... }
if (boolean_expression) { .... } else if (boolean_expression) {.... }
if (boolean_expression) { .... } else if (boolean_expression) {.... } else {....}
```

Useful Functions

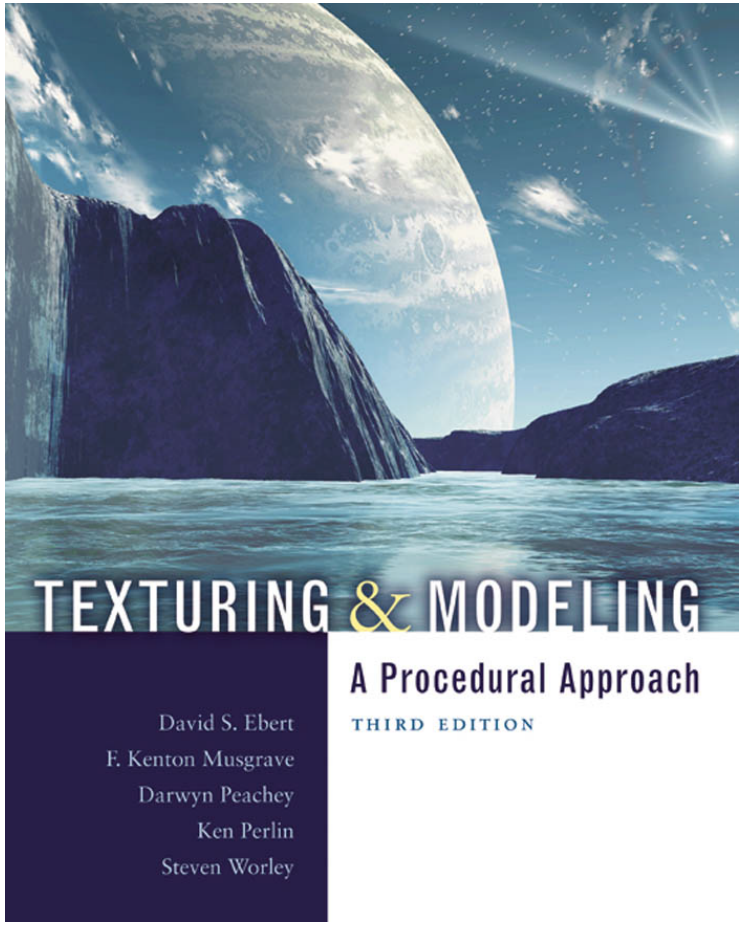
- resize(): similar to scale, but specify the size directly, e.g., resize([30,60,10]) sphere(r=10);
- mirror(): mirror the element on a plane through origin, argument is the normal vector of the plane, e.g., mirror([0,1,0]);
- hull(); create a hull from all objects that are inside, e.g., hull() {# translate([0,70,0]) circle(10); # circle(30); }
- minkowski(); takes one 2D shape and traces it around the edge of another 2D shape, e.g., minkowski() { cube([30,30,5]); # sphere(5);}





Further Reading in Procedural Techniques

- Texturing and Modeling - A Procedural Approach



That's All For Today

- Further Readings:
  - CSG in CGAL: http://doc.cgal.org/latest/Nef_3/index.html
  - *Procedural Modeling of Cities / Yoav Parish, Pascal Müller, Siggraph 2001*
  - *Procedural Modeling of Buildings / Müller et al, Siggraph 2006*
  - Converting 3D Furniture Models to Fabricable Parts and Connectors, Lau et al., Siggraph 2011