Constructive Solid Geometry
and
Procedural Modeling

Stelian Coros
Somewhat unrelated
Schedule for presentations

<table>
<thead>
<tr>
<th>February</th>
<th>3</th>
<th>5</th>
<th>10</th>
<th>12</th>
<th>17</th>
<th>19</th>
<th>24</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>March</td>
<td>3</td>
<td>5</td>
<td>10</td>
<td>12</td>
<td>17</td>
<td>19</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>April</td>
<td>2</td>
<td>7</td>
<td>9</td>
<td>14</td>
<td>16</td>
<td>21</td>
<td>23</td>
<td>28</td>
</tr>
</tbody>
</table>

Send me:
**ASAP:** 3 choices for dates + approximate topic (scheduling)
**1-2 weeks before your presentation:** list of papers you plan to talk about
**Day before each presentation:** 3 questions for one of the papers that will be discussed
Previous Lecture: Solid Modeling

- Represent solid interiors of objects
  - Voxels
  - Octrees
  - Tetrahedra
  - Distance Fields
Previous Lecture: From Surfaces to Voxels

- Ray casting
  - Trace a ray from each voxel center
  - Count intersections
    - Odd: inside
    - Even: outside
Real-life meshes

Self-intersections
Real-life meshes

Self-intersections

Nonmanifold edges
Real-life meshes

- Self-intersections
- Nonmanifold edges
- Open boundaries
Real-life meshes: output of human creativity, for better or worse
Robust Inside-Outside Segmentation using Generalized Winding Numbers

Alec Jacobson
Ladislav Kavan
Olga Sorkine-Hornung

ETH Zurich
University of Pennsylvania
ETH Zurich
Robust Inside-Outside Segmentation using Generalized Winding Numbers

- Main challenge - determine which points are inside of a shape, which are outside
If shape is watertight, \textit{winding number} is perfect measure of inside

- Winding number for a point in space:
  - how many times does the curve wind about the point
  Or, equivalently
  - Signed length of the curve projected on unit circle about the point

\[
\omega(p) = \frac{1}{2\pi} \oint_{c} d\theta
\]
If shape is watertight, *winding number* is perfect measure of inside

- Winding number for a point in space:
  - how many times does the curve wind about the point
  
  Or, equivalently

  - Signed length of the curve projected on unit circle about the point

\[
 w(p) = \frac{1}{2\pi} \oint_C d\theta
\]
Robust for: arbitrary topologies, self-intersections, overlaps, and multiple connected components

- Use orientation of curve to treat *insideness* as integer quantity
Winding number discretization (2D)

- Integral becomes sum of signed angle subtended by each edge

\[
w(p) = \frac{1}{2\pi} \oint_c d\theta
\]

\[
w(p) = \frac{1}{2\pi} \sum_{i=1}^{n} \theta_i
\]
Winding number discretization (3D)

- **Solid** angle subtended by each triangle

\[
w(p) = \frac{1}{4\pi} \int \int_S \sin(\phi) d\theta d\phi
\]

\[
w(p) = \frac{1}{4\pi} \sum_{f=1}^{m} \Omega_f
\]
From nice meshes to real-world meshes

- Winding number no longer an integer value

\[ w(p) = \frac{1}{2\pi} \oint_C d\theta \]

Gracefully tends toward perfect indicator as shape tends towards watertight
What if shape is self-intersecting? Non-manifold?

Normally smooth, jumps by \pm 1 across input facets
Sharp discontinuity across input eases precise segmentation
Winding number degrades gracefully
Winding number vs ray casting

1 ray
Winding number vs ray casting

3 rays
Winding number vs ray casting

7 rays
Winding number vs ray casting

15 rays
Winding number vs ray casting

31 rays
Winding number vs ray casting

63 rays
Winding number vs ray casting

127 rays
Winding number vs ray casting

511 rays
Winding number vs ray casting

2047 rays
Robust Inside-Outside Segmentation using Generalized Winding Numbers

Alec Jacobson
Ladislav Kavan
Olga Sorkine-Hornung

ETH Zurich
University of Pennsylvania
ETH Zurich
Geometric Representations

- Languages for describing shape
  - Boundary representations
    - Polygonal meshes
    - Subdivision surfaces
    - Implicit surfaces
  - Volumetric models
  - Parametric models
  - Procedural/generative models

Lower Level

Higher Level
Constructive Solid Geometry (CSG)

- A way of building complex objects from simple primitives using Boolean operations
Constructive Solid Geometry (CSG)

- Represent solid object as hierarchy of Boolean operations
- The Boolean operations are not evaluated
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
Booleans for Solids

Given overlapping shapes A and B:

- **Union**: Includes both A and B
- **Intersection**: Only the overlapping part of A and B
- **Subtraction**: A minus the intersection with B
How Can We Implement Boolean Operations?

- Use voxels/octrees/ADFs
  - We can convert from b-reps to voxels/DF and back
  - Process objects voxel by voxel
  - Issues?
How Can We Implement Boolean Operations?

- Directly: the hard way ...
  - You will not be asked to do this
- Commercial libraries/CAD tools
  - e.g., Parasolid, SolidWorks
- Open source libraries
  - e.g., CGAL, 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

## 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`
- `cell`
- `ln`
- `len`
- `log`
- `lookup`
- `min`
- `max`
- `pow`
- `sort`
- `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);`
**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 “;”
  - Thickness is 1mm
  - Use “[“ and “]” to pass multiple values
2D to 3D Extrusion

• Linear extrusion
  - Extrudes a 2D shape along the Z axis
    linear_extrude(height = 10, center = true, convexity = 10, twist = -100) translate([2, 0, 0]) circle(r = 1);

• Rotational extrusion
  - Revolves a 2D shape around the Z axis
    rotate_extrude($fn=200)
    polygon(points=[[0,0],[2,1],[1,2],[1,3],[3,4],[0,5]]);
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**
  - Color(“yellow”)
    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

```plaintext
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**

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

- **Default values**

```plaintext
gmodule 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

```plaintext
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);
}
```

```plaintext
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.,

```openSCAD
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++
Useful Functions

- 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 convex 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); }
The Plan For Today

- Constructive Solid Geometry (CSG)
  - Parametric models from simple primitives
- Procedural Modeling
The Plan For Today

• Constructive Solid Geometry (CSG)
  – Parametric models from simple primitives
• Procedural Modeling
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 → 11), (0 → 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
L-system Example 2: Fractal Plant

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

- Tree examples
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
  
  $0 (0.5) \rightarrow 1[0]0$
  
  $0 (0.5) \rightarrow 010$
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

\[
\begin{align*}
S & \rightarrow aSBC \\
S & \rightarrow aBC \\
CB & \rightarrow HB \\
HB & \rightarrow HC \\
HC & \rightarrow BC \\
aB & \rightarrow ab \\
bB & \rightarrow bb \\
bC & \rightarrow bc \\
cC & \rightarrow cc
\end{align*}
\]
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

  - \( A(x,y) \rightarrow A(1, y+1)B(x-2,3) \)
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/papers)
  - [http://algorithmicbotany.org/virtual_laboratory/](http://algorithmicbotany.org/virtual_laboratory/)
TreeSketch: Interactive Tree Modeling

http://vimeo.com/68195050
Procedural Modeling of Buildings

- Pompeii
Procedural Modeling of Buildings / Müller et al, Siggraph 2006
CityEngine

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
Pre-defined formal grammar used to analyze structure of 3D models
Example: 2D Cabinet
Examples of Production Rules

Production Rule 1

Production Rule 2
Examples of Production Rules

Production Rule 4

Production Rule 6

Production Rule 8
Sequence of Production Rules
All Production Rules

1. $S \rightarrow \nu - X - \nu$
2. $\nu - X - \nu \rightarrow \nu \rightarrow \nu$
3. $\nu \rightarrow \nu \rightarrow \nu$
4. $ht \rightarrow X \rightarrow -X - \nu - X - \nu$
5. $-X \rightarrow \nu \rightarrow \nu$
6. $-X \rightarrow \nu \rightarrow \nu$
7. $B \rightarrow \nu \rightarrow \nu$
8. $B \rightarrow \nu \rightarrow \nu$
9. $B \rightarrow \nu \rightarrow \nu$
10. $B \rightarrow \nu \rightarrow \nu$

$ht \rightarrow \nu \rightarrow \nu$

$X \rightarrow \nu \rightarrow \nu$

$B \rightarrow \nu \rightarrow \nu$

$B \rightarrow \nu \rightarrow \nu$

$B \rightarrow \nu \rightarrow \nu$
Formal Grammar for 2D Cabinets

\[ N = \{ S, B, X, Y \} \]

\[ \Sigma = \{ \text{hb, ht, v, ha, leg, wheel} \} \]

The language specifies a directed graph which represents parts and connectors.
Overview of algorithm
Overview of algorithm

Lexical Analysis:
Identify separate tokens
(i.e. primitive shapes) from model

Multiple valid options
Grammar-based Furniture Design

Converting 3D Furniture Models to Fabricatable Parts and Connectors

Manfred Lau, Akira Ohgawara, Jun Mitani, Takeo Igarashi

JST ERATO Igarashi Design Interface Project
University of Tsukuba The University of Tokyo
Procedural Modelling

Procedural Modeling of Structurally-Sound Masonry Buildings

Submission ID: 0105

[contains audio]
That’s All For Today