Constructive Solid Geometry and Procedural Modeling

Wojciech Matusik
CSAIL & EECS MIT
Previous Lecture: Solid Modeling

- Represent solid interiors of objects
- Voxels
- Octrees
- Tetrahedra
- Distance Fields
The Plan For Today

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

- A neat way to build complex objects from simple parts using Boolean operations
  - Very easy when ray tracing (6.837)
  - Not so easy when not ray tracing
Constructive Solid Geometry (CSG) in CAD

- Interactive modeling programs
  - Intuitive way to design objects
Constructive Solid Geometry (CSG)

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

Given overlapping shapes A and B:

Union                   Intersection            Subtraction

Should only “count” overlap region once!
How Can We Render CSG?

4 Cases

- Points on A, Outside of B
- Points on B, Inside of A
- Points on A, Inside of B
- Points on B, Outside of A

Union

Intersection

Subtraction
How Can We Render CSG?

- Use ray casting
Ray Casting: Collect Intersections

Each ray processed separately!

Union

Intersection

Subtraction
Ray Casting CSG

1. Test "inside" intersections:
   - Find intersections with A, test if they are inside/outside B
   - Find intersections with B, test if they are inside/outside A

This would certainly work, but would need to determine if points are inside solids... :-(
Ray Casting CSG

1. Test "inside" intersections:
   • Find intersections with A and test if they are inside/outside B
   • Find intersections with B, test if they are inside/outside A

2. Overlapping intervals:
   • Find the intervals of "inside" along the ray for A and B
   • How? Just keep an “entry” / “exit” bit for each intersection
     • Easy to determine from intersection normal and ray direction
   • Compute union/intersection/subtraction of the intervals
Problem reduces to 1D for each ray

2. Overlapping intervals:
   • Find the intervals of "inside" along the ray for A and B
   • How? Just keep an “entry” / “exit” bit for each intersection
   • Easy to determine from intersection normal and ray direction
   • Compute union/intersection/subtraction of the intervals
Rendering CSG is Easy with Ray Casting...

- ...but very hard if you actually try to compute an explicit representation of the resulting surface as a triangle mesh

- In principle very simple, but floating point numbers are not exact
  - E.g., points do not lie exactly on planes...
  - Computing the intersection A vs B is not necessarily the same as B vs A...
  - The line that results from intersecting two planes does not necessarily lie on either plane...
  - etc., etc.
How Can We Implement Boolean Operations?

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

![Diagram showing boolean operations using voxels](image-url)
How Can We Implement Boolean Operations?

• The hard way ...
• 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, ...
Boolean Operations in CGAL

• Based on Nef Polyhedra

• Nef Polyhedron:
  - generated from a finite number of halfspaces by complement and intersection operation
    \[ AX + BY + CZ + D > 0 \]
  - union and difference can be generated from intersection and complement
Boolean Operations in CGAL

- Evaluate a CSG-tree with halfspaces as primitives and convert it into a B-rep representation

```cpp
Nef_polyhedron N1(Plane_3( 1, 0, 0,-1));
Nef_polyhedron N2(Plane_3(-1, 0, 0,-1));
Nef_polyhedron N3(Plane_3( 0, 1, 0,-1));
Nef_polyhedron N4(Plane_3( 0,-1, 0,-1));
Nef_polyhedron N5(Plane_3( 0, 0, 1,-1));
Nef_polyhedron N6(Plane_3( 0, 0,-1,-1));

Nef_polyhedron I1(!N1 + !N2);  // open slice in yz-plane
Nef_polyhedron I2(N3 - !N4);   // closed slice in xz-plane
Nef_polyhedron I3(N5 ^ N6);    // open slice in yz-plane
Nef_polyhedron Cubel(I2 * !I1);
Cubel *= !I3;
Nef_polyhedron Cube2 = N1 * N2 * N3 * N4 * N5 * N6;
```

CSG Examples
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
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 start symbol: $S$

Generates a set of finite-length sequences of symbols by recursively applying production rules starting with $S$
Formal Grammars and Languages

- Chomsky’s four types of grammars

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Languages</th>
<th>Automaton</th>
<th>Production rules (constraints)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-0</td>
<td>Recursively enumerable</td>
<td>Turing machine</td>
<td>$\alpha \rightarrow \beta$ (no restrictions)</td>
</tr>
<tr>
<td>Type-1</td>
<td>Context-sensitive</td>
<td>Linear-bounded non-deterministic Turing machine</td>
<td>$\alpha A\beta \rightarrow \alpha \gamma \beta$</td>
</tr>
<tr>
<td>Type-2</td>
<td>Context-free</td>
<td>Non-deterministic pushdown automaton</td>
<td>$A \rightarrow \gamma$</td>
</tr>
<tr>
<td>Type-3</td>
<td>Regular</td>
<td>Finite state automaton</td>
<td>$A \rightarrow a$ and $A \rightarrow aB$</td>
</tr>
</tbody>
</table>

Recursively enumerable

Context-sensitive

Context-free

Regular
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 1: Algae

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

\[\begin{align*}
n = 0 & : A \\
n = 1 & : AB \\
n = 2 & : ABA \\
n = 3 & : ABAAB \\
n = 4 & : ABAABABA \\
n = 5 & : ABAABABAABAAB \end{align*}\]
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

![Axiom](image1)
![First recursion](image2)
![Second recursion](image3)
![Third recursion](image4)
![Fourth recursion](image5)
![Seventh recursion, scaled down ten times](image6)
L-system Example 3: Fractal Plant

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

• Tree examples

<table>
<thead>
<tr>
<th></th>
<th>n=5, δ=25.7°</th>
<th>n=5, δ=20°</th>
<th>n=4, δ=22.5°</th>
<th>n=7, δ=20°</th>
<th>n=7, δ=25.7°</th>
<th>n=5, δ=22.5°</th>
</tr>
</thead>
</table>
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

\[
\begin{align*}
S & \rightarrow aSBC \\
S & \rightarrow aBC \\
CB & \rightarrow HB \\
HB & \rightarrow HC \\
HC & \rightarrow BC \\
aB & \rightarrow ab \\
bB & \rightarrow bb \\
bC & \rightarrow bc \\
cC & \rightarrow cc
\end{align*}
\]
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
  - \[ A(x,y): x = 0 \rightarrow A(1, y+1)B(2,3) \]
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 Cities

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

- Pompeii
Procedural Modeling of Buildings

- Modern architecture
Procedural Modeling of Buildings

- Mayan architecture
Procedural Modeling of Buildings

Pascal Müller
Peter Wonka
Simon Haegler
Anreas Ulmer
Luc Van Gool
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

Pre-defined formal grammar to analyze structure

Separate parts and connectors

Formal grammar

S → X

ht

B
Representation of 2D Cabinets

Example 2D Cabinet  Corresponding Graph  Positioning of Parts
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

1. $S \rightarrow v - X - v \\
   \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
   \quad B \quad B \quad B \quad B$

2. $v - X - v \quad \rightarrow \quad v - ha - v \\
   \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
   \quad v \quad v \quad v \quad v$

3. $v - Y - v \quad \rightarrow \quad v - ha - v \\
   \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
   \quad v \quad v \quad v \quad v$

4. $-X- \quad \rightarrow \quad -X- -v- -X- \\
   \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
   \quad B \quad B \quad B \quad B$

5. $-X- \quad \rightarrow \quad \varepsilon \\
   \quad \downarrow \quad \downarrow \\
   \quad v \quad v$

6. $-Y- \quad \rightarrow \quad \varepsilon \\
   \quad \downarrow \quad \downarrow \\
   \quad v \quad v$

7. $B \quad \rightarrow \quad hb \\
   \quad \downarrow \quad \downarrow \\
   \quad B \quad B$

8. $B \quad \rightarrow \quad wheel \quad wheel \\
   \quad \quad \downarrow \quad \quad \downarrow \\
   \quad B \quad B$

9. $B \quad \rightarrow \quad leg \quad leg \\
   \quad \downarrow \quad \downarrow \\
   \quad B \quad B$
Formal Grammar for 2D Cabinets

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

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

\[ P : \text{Set of Production Rules} \]

Non-terminal Symbols
- Collection of Parts

Terminal Symbols
- Separate Parts

The language specifies a directed graph, and each graph represents parts and connectors
Results: IKEA ALVE Cabinet

Input: 3D model (Google Warehouse) IKEA ALVE cabinet
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)**

```
sphere(r=20) ;
```
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]);`
• Union
• Intersection
• Difference
• Example:

```c
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**

  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

```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);
}

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.,:

```plaintext
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

- **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

• Readings:
  - *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