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
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
    • \( 1/(r^2+e); 1/(r^2)^2; \text{gaussian} \)
  • Original metaballs did not have \( c_i \)
CSG Data Structure

- Stored in a Binary Tree

Data structure:

```
subtract
  intersect
    blue circle
    red cube
  union
    green union
      green cylinder
      green cone
```
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”

```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);
  }
```
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: 111111[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

```
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
```

Axiom | First recursion | Second recursion | Third recursion | Fourth recursion | Seventh recursion, scaled down ten times
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/)
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
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,

\[
\begin{bmatrix}
\max(x_{1,1}, x_{2,1}, \ldots, x_{n,1}) \\
\cdots \\
\max(x_{1,d}, x_{2,d}, \ldots, x_{n,d})
\end{bmatrix}
\]

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
      - choose eps, and K big enough
    - then there is some g(S) of this form st |f(S) - g(S)| < eps
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.

Qi et al 17
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.
This is a general summarization procedure

- Point clouds aren’t just 3D points
- Examples:
  - (x, y, z, r, g, b)
  - (x, y, z, 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