Geometric modeling - Motion

How do we represent objects?
Solid Modeling

Do the lines define a solid cube?
Wish List

- Large enough domain
- Unambiguous
- Unique
- Accurate
- Impossible to create invalid representations
- Easy to create and maintain
- Closure under certain operations
- Compact
- Efficient
Regularized Boolean Set Operations

Will applying Boolean set operations to solid objects yield a solid object?
Example

(a)  (b)  (c)  

(d)  (e)
Definitions

- **Boundary points** - distance from object and object's complement is 0
- **Closed set** - contains all its boundary points
- **Open set** - contains none
- **Closure** - union of a set with its boundary
- **Regularization** - the closure of the set's interior points
- **Regular set** - equals to its regularization
Example

object

interior

closure

regularization
Regularized Boolean Set Operations

\[ A \text{ op}^* B = \text{closure}(\text{interior}(A \text{ op} B)) \]
Roadmap

1. Primitive instancing
2. Sweep representations
3. Constructive solid geometry (CSG)
4. B-Reps, Polyhedron
5. Spatial-partitioning representations
Primitive Instancing

The modeling system defines a set of primitive 3D solid shapes that are relevant to the application area.
Sweep Representations

Sweeping an object along a trajectory through space defines a new object, called a sweep

- Translational sweep, extrusion
- Rotational sweep
- General sweep
Constructive Solid Geometry (CSG)

- Combine simple primitives by means of regularized Boolean set operators
- An object is stored as a tree:
  - Leaves - primitives
  - Internal nodes - operators (regularized boolean or transformations)
Example
Example
Boundary Representations (B-Reps)

- **B-reps** - describe an object in terms of its surface boundaries: usually vertices, edges, faces
Polygonal mesh (polygon soup)
2-manifold

Every point on it has some arbitrary small neighborhood of points that can be considered topologically as a disk in the plane

Not 2-manifold
Polyhedron

A finite collection of planar, bounded, convex polygonal faces such that:
1. The faces intersect “properly”
2. The neighborhood of every point is topologically an open disk
3. The surface is connected
Euler Formula

For genus zero polyhedra:  \( V - E + F = 2 \)
Polyhedron Representation

- List of faces
- Indexed face list
- Winged-Edge representation
Winged-Edge representation

- An edge is represented by pointers to
  - 2 vertices
  - 2 faces
  - 4 edges
- A edge/face is represented by a pointer to an edge
Spatial-Partitioning Representations

1. Cell decomposition
2. Spatial-occupancy enumeration (voxels)
3. Octrees
4. Binary space-partitioning trees
Voxels
Example

Visible man, GE
Octree

- Divide-and-conquer approach
- It is an extension of a quadtree
Finding Neighbors

Algorithm North-Neighbor (v, T)
if v = root(T)
    return nil
if v = SW-child of parent(v)
    return NW-child of parent(v)
if v = SE-child of parent(v)
    return NE-child of parent(v)
μ = North-Neighbor (parent(v), T)
if μ = nil or μ is a leaf
    return μ
else
    if v = NW-child of parent(v)
        return SW-child of μ
    else
        return SE-child of μ
Balanced Quadtrees
Algorithm for Balancing Quadtrees

Algorithm BalanceQuadtree(T)
  Insert all leaves into list L
  while L is not empty
    remove leaf $\mu$ from L
    if $\sigma(\mu)$ has to be split
      make $\mu$ into an internal node
      insert the 4 new leaves into L
    if $\sigma(\mu)$ now has neighbors that need to be split
      insert them into L
Octrees

How do we perform boolean operations?
Binary Space Partitioning (BSP) Trees

Recursively split the space (plane) with a plane (line).
2D BSP Creation

Algorithm 2D-BSP

if card(S) <= 1
    Create a tree T of a single node,
    Store S explicitly;
    return T
else /* use l(s1) as splitting line */
    S⁺ ← \{s ∩ l(s1)⁺ : S ⊊ s\};
    T⁺ ← 2D-BSP(S⁺);
    S⁻ ← \{s ∩ l(s1)⁻ : S ⊊ s\};
    T⁻ ← 2D-BSP(S⁻);
    Create T with root at v, left subtree is T⁻, right is T⁺
    S(v) = \{S ⊊ s : s ⊆ l(s1)\}
    return T
3D BSP Creation

Algorithm 3D-BSP

if card(S) <= 1
    Create a tree T of a single node,
    Store S explicitly;
    return T
else /* use h(s1) as splitting line */
    S⁺ ← \{t \cap h(t1): S \ni t\};
    T⁺ ← 3D-BSP(S⁺);
    S⁻ ← \{t \cap h(t1): S \ni t\};
    T⁻ ← 3D-BSP(S⁻);
    Create T with root at v, left subtree is T⁻, right is T⁺
    S(v) = \{T \ni t: t \subset h(t1)\}
    return T