Computational Design + Fabrication: 3D Design

Jonathan Bachrach
EECS UC Berkeley
September 24, 2015
Today

- News
- 3D Design
- 3D Geometry
- Paper Review
- Lab 2 due Thursday
- Section tomorrow 2-3p in Soda 373
- Jacobs 3D printer training
  https://bcourses.berkeley.edu/courses/1353091
- safety glasses are to be worn all the time when using diwire machine
- students are responsible for supplying their own safety glasses
- students must put away all tools when done
- bricks and legos
- 3d printing
Motivation

- complexity is free
- less waste
- no assembly required
- less skill
- examples
- solid modeling
- digifab support
3D Printed Lamp

by 74fdc.wordpress
Atom 3D Printed Guitar

by cubify
Bracelet by Nervous Systems
Insect Model

by Klaus Leitl
Intake Manifold

by Kevin Gautier for Formula SAE
Orthodontic Retainer

by Michael Lyons et al via FormLabs
Printed Fabric

by Designer Jiri Evenhuis, in collaboration with Janne Kyttanen of Freedom of Creation
Sketch Figurine

by Crayon Creations
by Chinese Anonymous
Pinhole Camera

by Anonymous
Tortoise Prosthesis

by Roger Henry
Ice

by TBWA/Hakuhodo
by Anonymous
by mediated matter group at mit – photo by andy ryan
Tiled 3DP Bricks

picoroco block in sand

shed out of blocks

by Emerging Objects
- digital models of physical objects
- represent interiors of objects
Boundary Representations (B-Reps)

- represents shapes using their limits
- using connected surface elements
- boundary between solid and non-solid
Pros
- flexible
- wide spread

Cons
- ill-defined solids (not closed under ops)
Mesh Representation

main topological components are:
- face – bounded portion of a surface
- edge – bounded piece of a curve
- vertex – lies at a point

other elements are:
- shell – connected set of faces
- loop – circuit of edges bounding a face
Winged-Edge Representation

Winged-Edge Meshes

<table>
<thead>
<tr>
<th>Face List</th>
<th>Edge List</th>
<th>Vertex List</th>
</tr>
</thead>
<tbody>
<tr>
<td>f0</td>
<td>e0</td>
<td>v0</td>
</tr>
<tr>
<td>f1</td>
<td>e1</td>
<td>v1</td>
</tr>
<tr>
<td>f2</td>
<td>e2</td>
<td>v2</td>
</tr>
<tr>
<td>f3</td>
<td>e3</td>
<td>v3</td>
</tr>
<tr>
<td>f4</td>
<td>e4</td>
<td>v4</td>
</tr>
<tr>
<td>f5</td>
<td>e5</td>
<td>v5</td>
</tr>
<tr>
<td>f6</td>
<td>e6</td>
<td>v6</td>
</tr>
<tr>
<td>f7</td>
<td>e7</td>
<td>v7</td>
</tr>
<tr>
<td>f8</td>
<td>e8</td>
<td>v8</td>
</tr>
<tr>
<td>f9</td>
<td>e9</td>
<td>v9</td>
</tr>
<tr>
<td>f10</td>
<td>e10</td>
<td></td>
</tr>
<tr>
<td>f11</td>
<td>e11</td>
<td></td>
</tr>
<tr>
<td>f12</td>
<td>e12</td>
<td></td>
</tr>
<tr>
<td>f13</td>
<td>e13</td>
<td></td>
</tr>
<tr>
<td>f14</td>
<td>e14</td>
<td></td>
</tr>
<tr>
<td>f15</td>
<td>e15</td>
<td></td>
</tr>
<tr>
<td>f16</td>
<td>e16</td>
<td></td>
</tr>
<tr>
<td>f17</td>
<td>e17</td>
<td></td>
</tr>
<tr>
<td>f18</td>
<td>e18</td>
<td></td>
</tr>
<tr>
<td>f19</td>
<td>e19</td>
<td></td>
</tr>
<tr>
<td>f20</td>
<td>e20</td>
<td></td>
</tr>
<tr>
<td>f21</td>
<td>e21</td>
<td></td>
</tr>
<tr>
<td>f22</td>
<td>e22</td>
<td></td>
</tr>
<tr>
<td>f23</td>
<td>e23</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. Winged-edge meshes

Winged Edge Structure
Half-Edge Representation

- Each **vertex** references one outgoing halfedge, i.e. a halfedge that starts at this vertex (1).
- Each **face** references one of the halfedges bounding it (2).
- Each **halfedge** provides a handle to
  - the vertex it points to (3),
  - the face it belongs to (4)
  - the next halfedge inside the face (ordered counter-clockwise) (5),
  - the opposite halfedge (6),
  - (optionally: the previous halfedge in the face (7)).

[Diagram of half-edge representation]

www.openmesh.org
represent interior
Why Volumetric Representations?

- scanning produces solids
- some apps require solids
- algorithms require solids
- some operations easier with solids
Some Volumetric Representations

- discrete volume representations
- implicit representations
- overlay grid on solid
- represent grid cells called voxels
can store solid properties in voxels such as

- occupancy
- color
- density

voxel3d
- $O(n^3)$ voxels
- for example 1 billion voxels in $1000^3$
- like image processing
- resampling / resizing
- examples blur, sharpen, edge detection, ...
Voxel Visualization

- ray casting
- slicing
- isosurfaces
Conversion from Voxels to Surfaces

- marching cubes

Lorenson
project rays and count
Isosurfaces

- slices at equal values

3ds max
Conversion from Surfaces to Voxels

- ray casting
- rasterization
Voxels Pros / Cons

pros
- simple and intuitive
- easy acquisition

cons
- approximate
- not affine invariant
- large
- slow to display
what is appropriate resolution?
like photoshop
Octree Representation

- hierarchical representation
- use detail where needed
- smaller

[Diagram of an octree structure]
Octree Results

- construct tree based on uniformity
- voxel techniques apply
- operations, display
time varying

space efficient

simulation
function takes point says

- in or out
- distance to closest point
primitives are half spaces
solids are operations on half spaces
Half Space Composition
- C++ library
- B-Rep also
pros

- closed under set operations
- builds boundary

cons

- curves approximated with lots of half spaces
Distance Fields

- distance to closest object for every point in space
- zero on boundary
- negative inside
- positive outside

perry + frisken
3D visualization of distance field of R

perry + frisken
 Iso Surfaces

- surfaces with equal distances
- zero iso surface is boundary
■ defined everywhere
■ easy inside/outside test
■ gradients of field provide useful information
  ■ on boundary gradient is surface normal
  ■ off boundary gradient is direction to closest boundary
Distance Field Operations

fast and simple operations:

- $\text{dist}(A \cap B) = \min(\text{dist}(A), \text{dist}(B))$
- $\text{dist}(A \cup B) = \max(\text{dist}(A), \text{dist}(B))$
- $\text{dist}(A - B) = \min(\text{dist}(A), -\text{dist}(B))$

```cpp
float opU(float d1, float d2) {
    return min(d1,d2);
}

float opS(float d1, float d2) {
    return max(-d1,d2);
}

float opI(float d1, float d2) {
    return max(d1,d2);
}
```
sphere \( \text{dist}(p) = \sqrt{(p - c)^2} \)
- regularly sampled
- octree
- adaptively sampled
use trilinear reconstruction

2D shape with sampled distances to the surface
Regularly sampled distance values
2D distance field

perry + frisken
Trilinear Interpolation

\[ P_1(x) = p_{000}(1-x)(1-y)(1-z) + p_{100}x(1-y)(1-z) + \\
p_{010}(1-x)y(1-z) + p_{001}(1-x)(1-y)z + p_{101}x(1-y)z + \\
p_{011}(1-x)yz + p_{110} xy(1-z) + p_{111}xyz \]

Figure 2. Trilinear interpolation

hagan + braley
- insufficient sampling results in aliasing
- excess sampling requires excess memory

equation 2D shape with sampled distances to the surface

Regularly sampled distance values

2D distance field

perry + frisken
still have to decide leaf resolution

perry + frisken
Adaptively Sampled

- stop when can reconstruct with sufficient accuracy
- use test candidates

Test to trivially determine if a cell is interior or exterior

19 test points to determine cell error

perry + frisken
Adaptively Sampled Comparison

23,573 cells (3-color)  1713 cells (ADF)
perry + frisken
- distance fields in C++
- optimized with octree and interval analysis
advantages

- functional + compositional
- blending
- powerful transformations
- unlimited precision
- iso surfaces – milling
- gradients
- scanning

disadvantages

- slow to render
- may be hard to get mesh out
- too mathematical
demo "cube(4)"
demo "blend(cube(4),sphere(4),3.5)"
demo "(0.1*(sin(4*x)+sin(4*y)+sin(4*z)))+cube(3)"
demo "xrevolve(square(4))"
demo "zrot(z,pyramid(-4,4,-4,4,-4,4))"
demo "mag1(6,lettercube(4))"
Kokopelli

by matt keeter
more general and abstract
ready for any fabrication method
pretty easy to convert
- PolyMesh
- Transformations
- Quaternions
- 3D -> 2D -> 3D
PolyMesh

- collection of meshes (or sea of polygons)
  - points
  - indices (faces)
- standard geometry operations
  - transformations
  - hull, bounds, ...
- constructors
  - load from stl
  - points + indices
  - generator from solidpython
Transformations

- create from numpy arrays
- constructors in transformation.py for basics
- composition can use compose or *= per docs
Rotation Representations

- euler angles – [rx,ry,rz]
- 3x3 matrix
- 4x4 matrix
- quaternion – [s,x,y,z]
Gimbal Lock Problem

- 3 euler angle rep can get locked when moving between angles
- two out of three gimbals are in same plane
- lose one degree of freedom

no gimbal lock  
gimbal lock

by MathsPoetry
Quaternions

- introduced by Hamilton in 1843
- succinct representation for rotations
- good for interpolation with no gimbal lock problem
- 4x4/3x3 matrix <-> quaternions
- provide extra degree of freedom
- avoids gimbal lock
- can smoothly and straightforwardly move between any rotations
- multiplication
- conjugate
- inverse
- interpolation – slerp
- conversions to/from other rotation representations
- shadow
- slice
- height
- scale
- rotate
next time

- 3D CNC
- 3D Rationalization
- 3D Joinery
- 3D Validation
References

- *Mesh Basics* by Dr. Ching-Kuang Shene

- *Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics* by Frisken + Perry + Rockwood + Jones

- *Quaternions* by Ken Shoemake

- *Kokopelli* by Matt Keeter
  http://www.mattkeeter.com/projects/kokopelli/

- *Antimony* by Matt Keeter
  http://www.mattkeeter.com/projects/antimony/