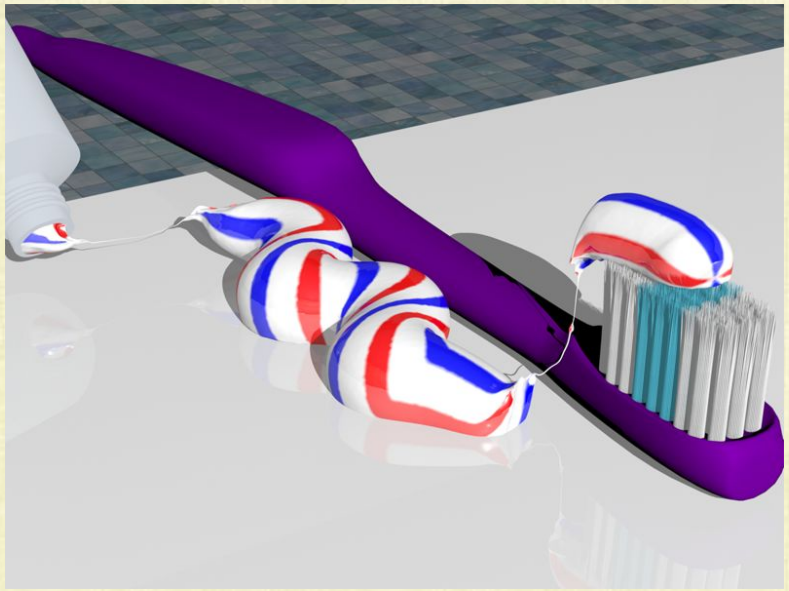
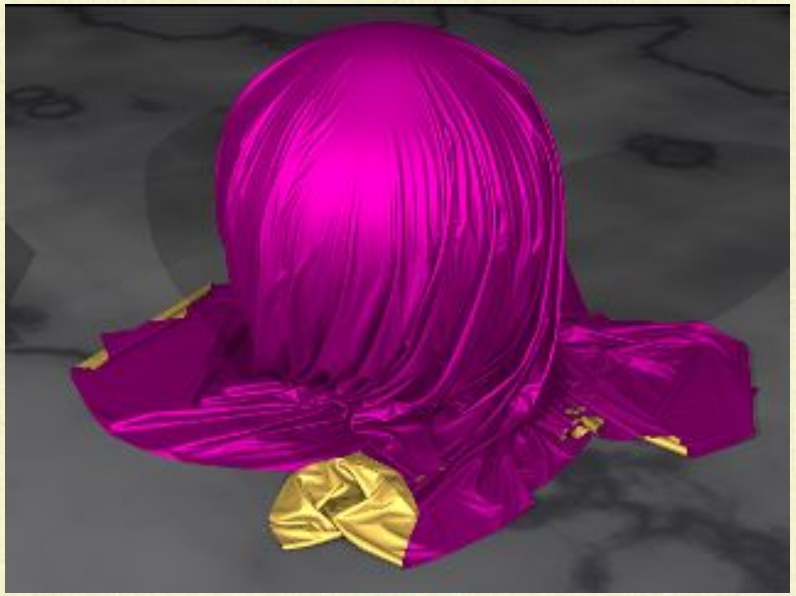
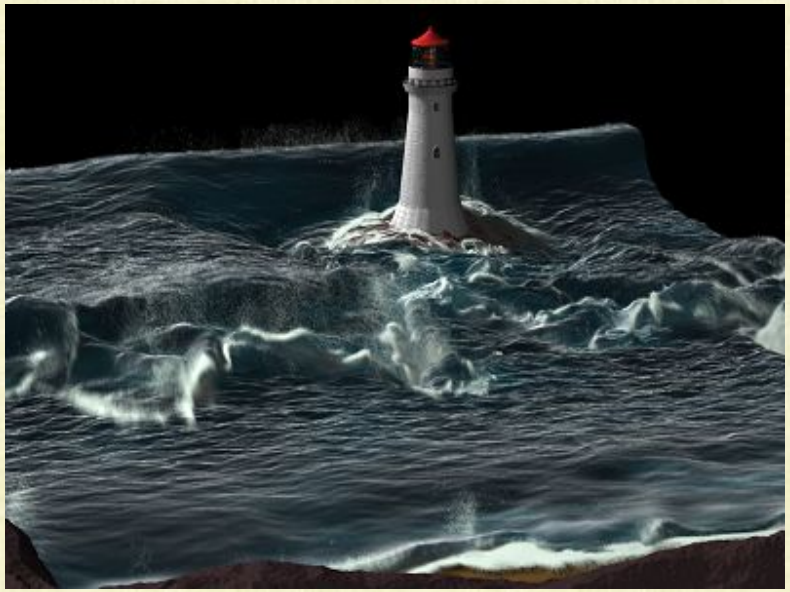
More Geometric Modeling



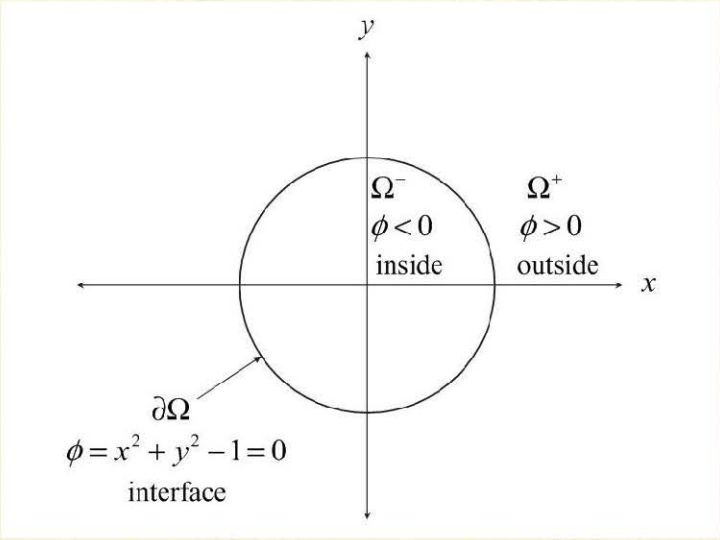


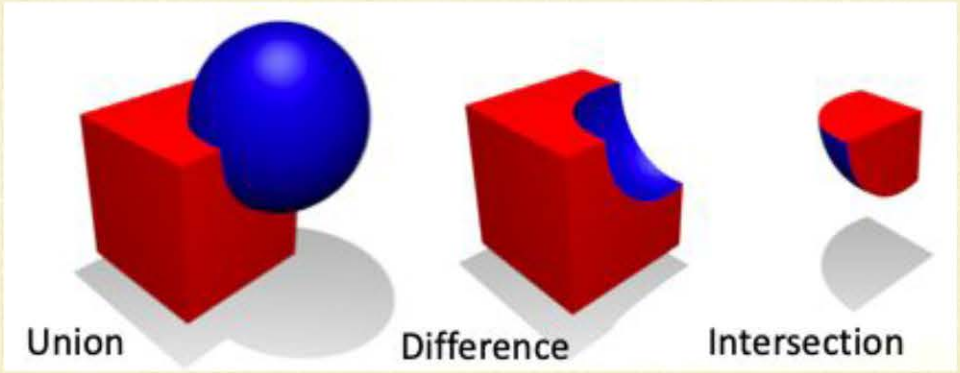


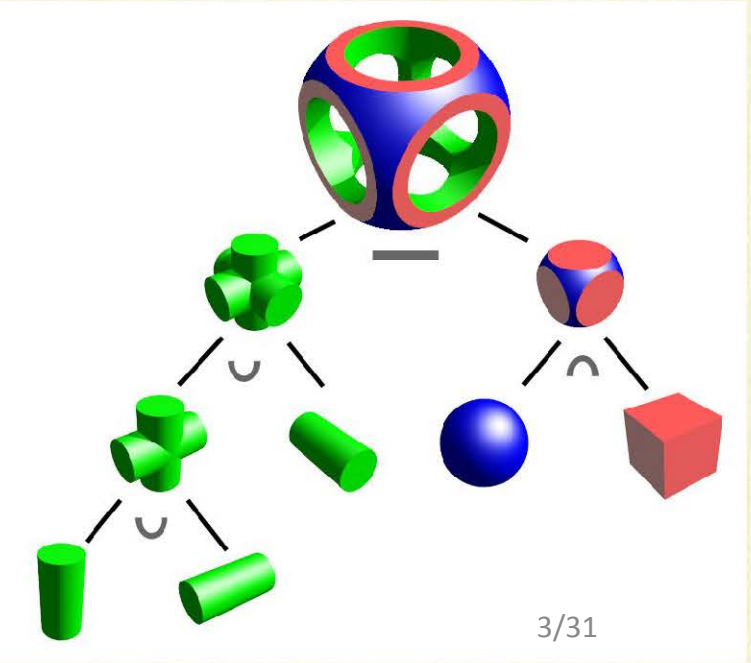
Implicit Surfaces

Implicit Surfaces

- Implicit surfaces define a function $\phi(x)$ over the entire 3D space ($x \in R^3$)
- The inside region Ω^- is defined by $\phi(x) < 0$, the outside region Ω^+ is defined by $\phi(x) > 0$, and the surface itself $\partial\Omega$ is defined by the isocontour where $\phi(x) = 0$
  - We have already seen planes/spheres (and lines/rays/circles in 2D) defined by implicit surfaces
- Easy to check if a point is inside/outside, simply by evaluating ϕ at the point
- Easy to do Constructive Solid Geometry (CSG) operations (union, difference, intersection, etc.)
- Easy to ray trace implicit objects!

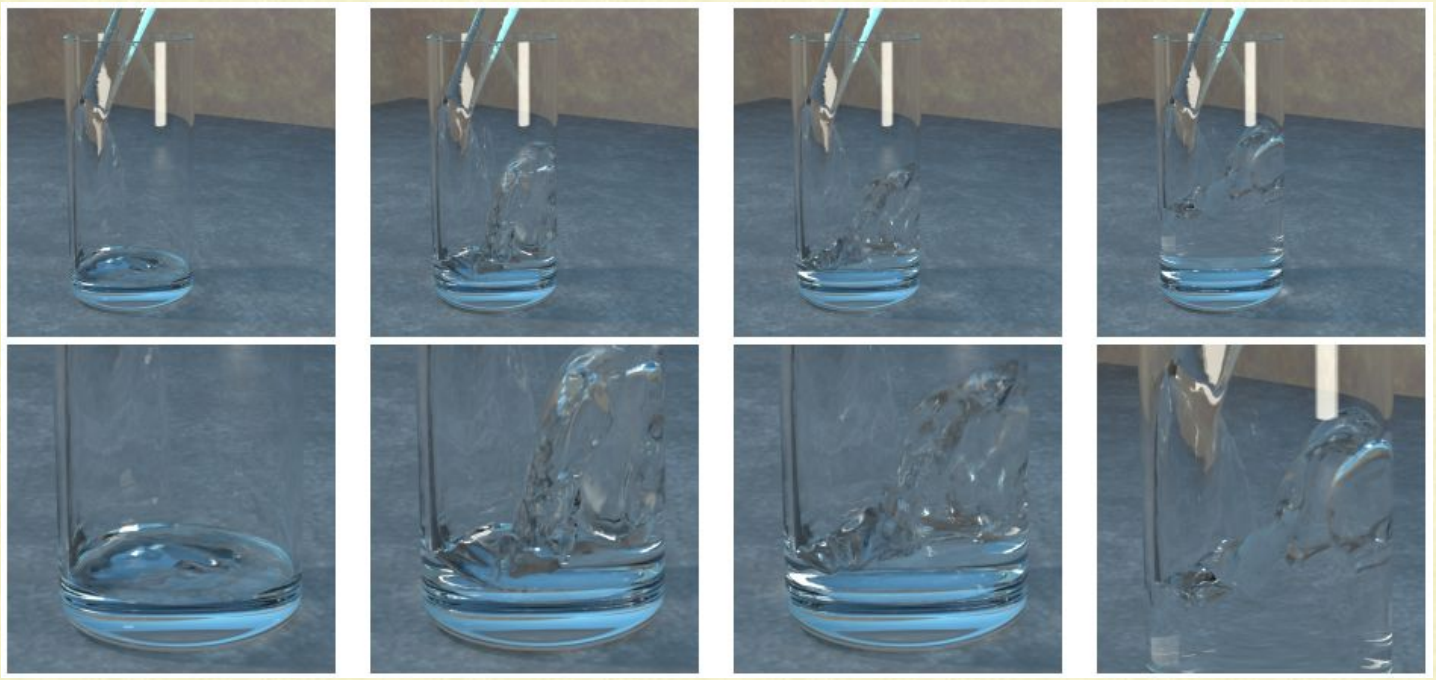


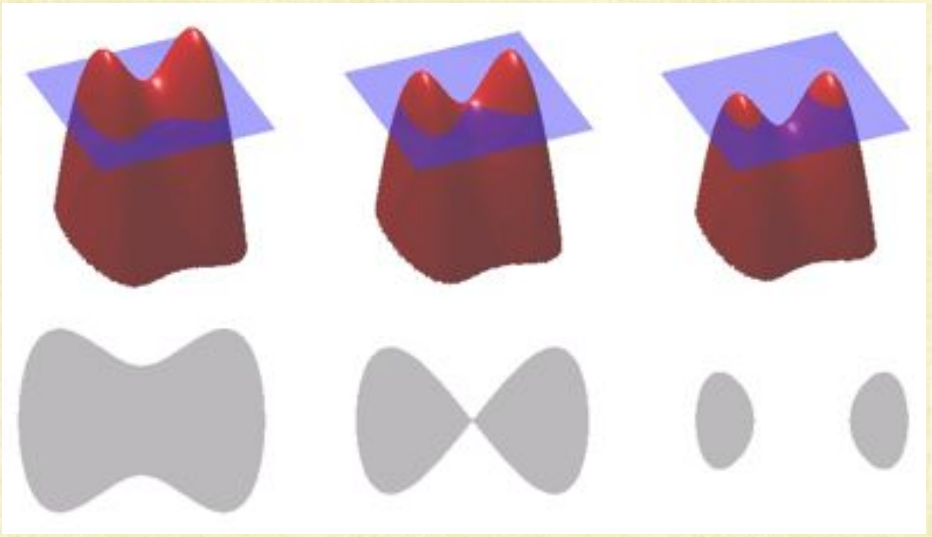


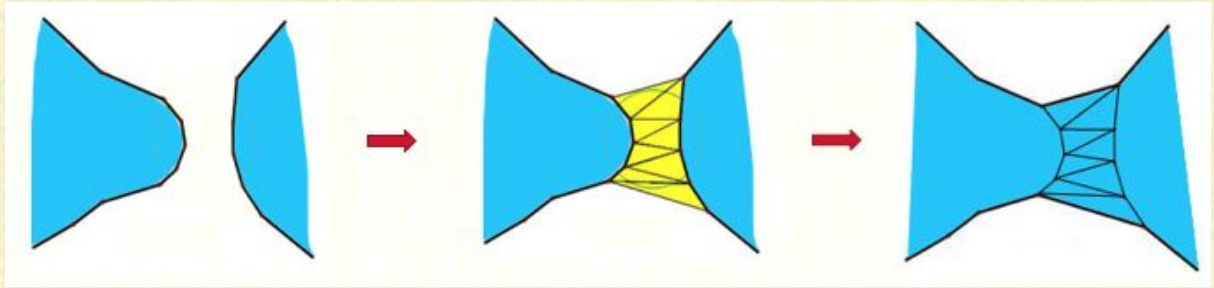


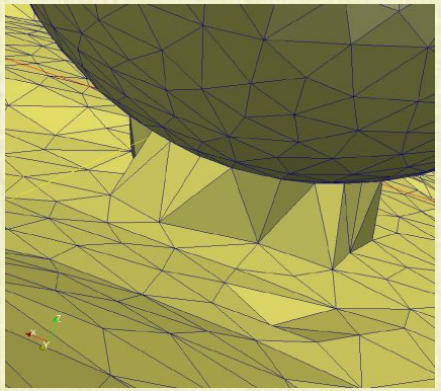
Topological Change

- Greatly superior to triangle meshes for topological change!



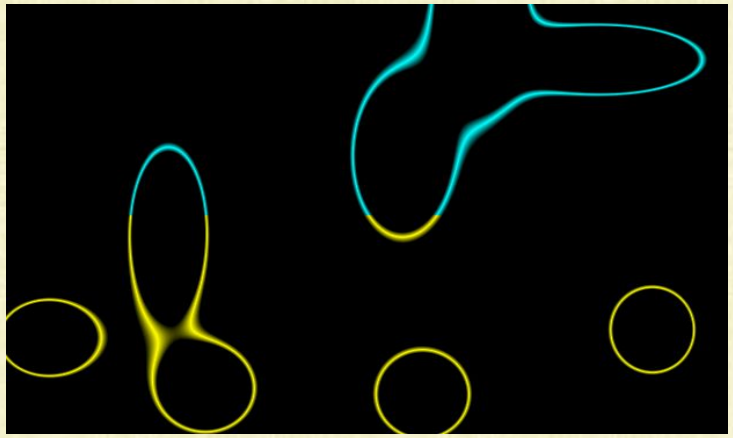


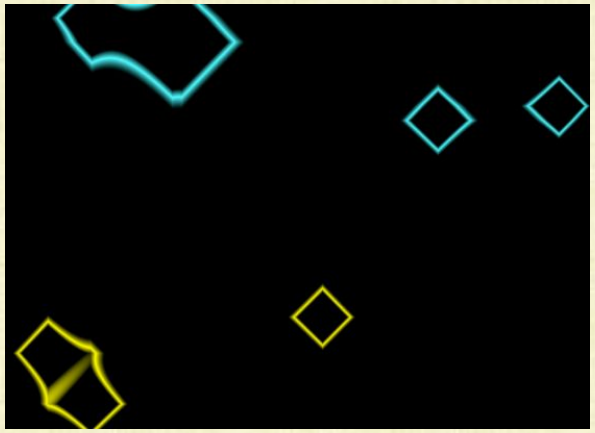


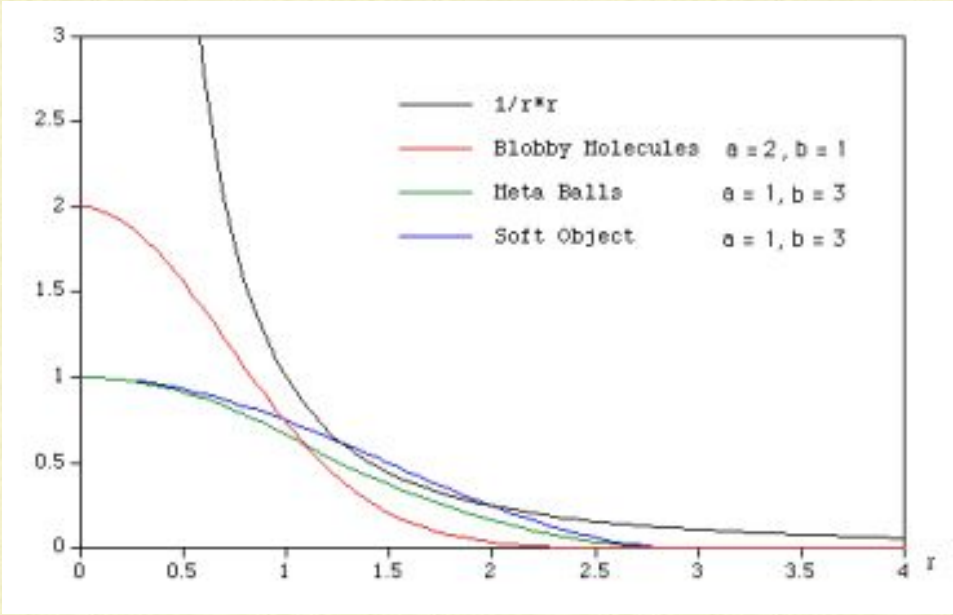


Blobbies

- Each blob is defined as a density function around a particle
- For each pixel, the aggregate function is created by summing the density function values of every blob
  - Blob kernels can be: 2D ellipses, 2D diamonds, 3D spheres, etc.
- Blobbies were proposed contemporaneously with Metaballs (in Japan) and Soft objects (in Canada and New Zealand)
  - Slightly different density kernel functions

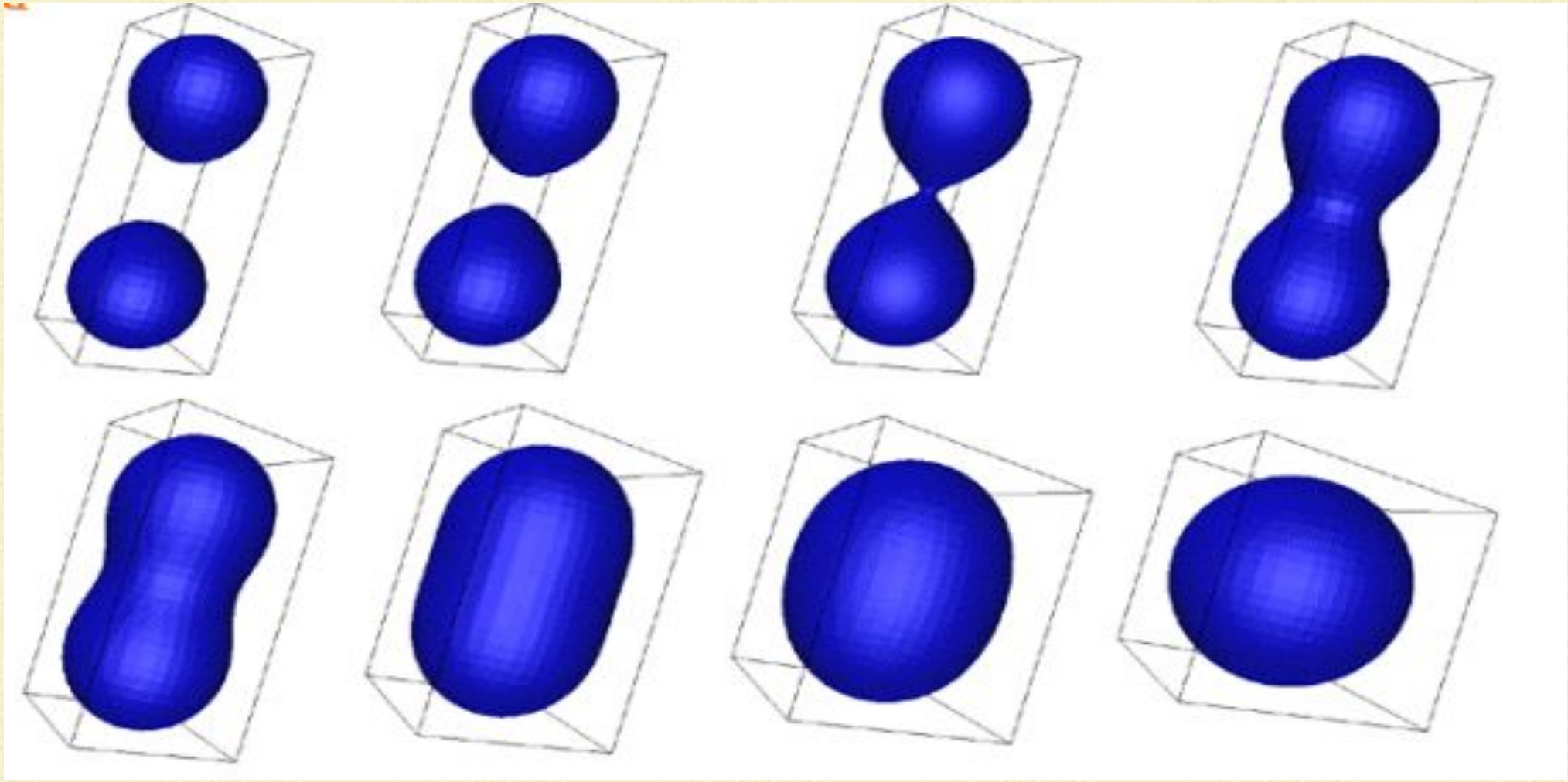




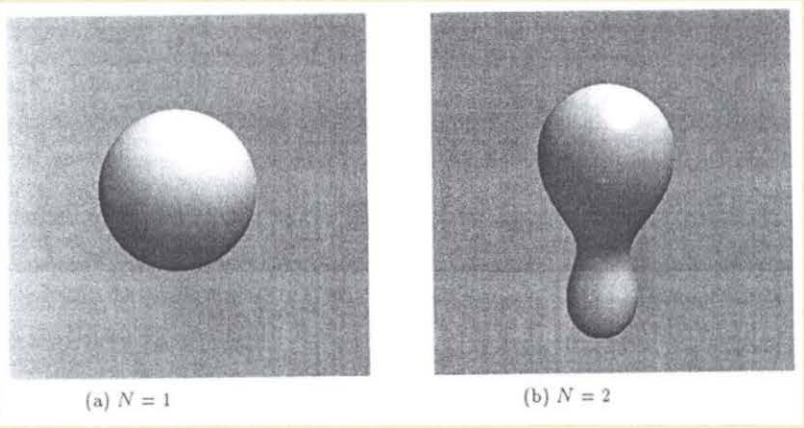


Topological Change

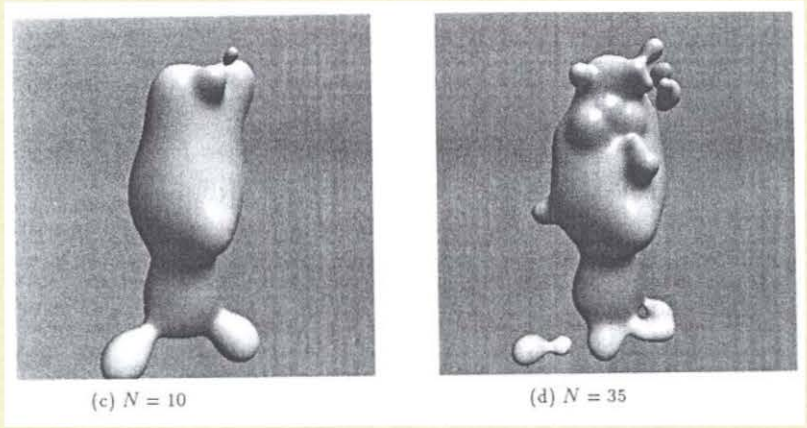
- Two blobby particles easily merging into one blob



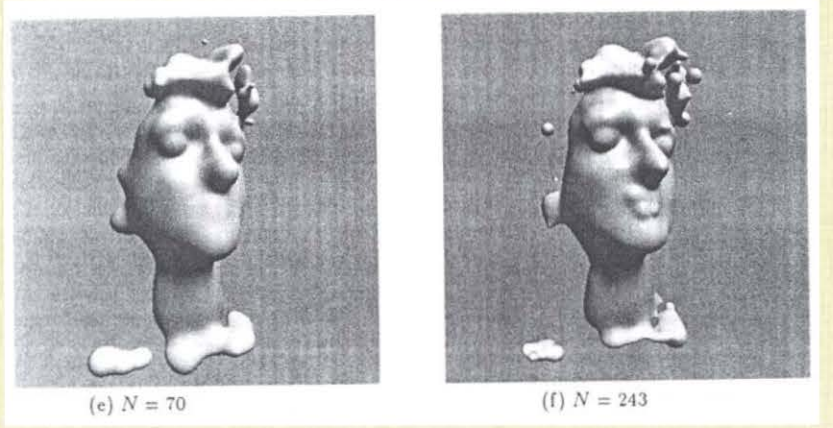
Blobby Modeling



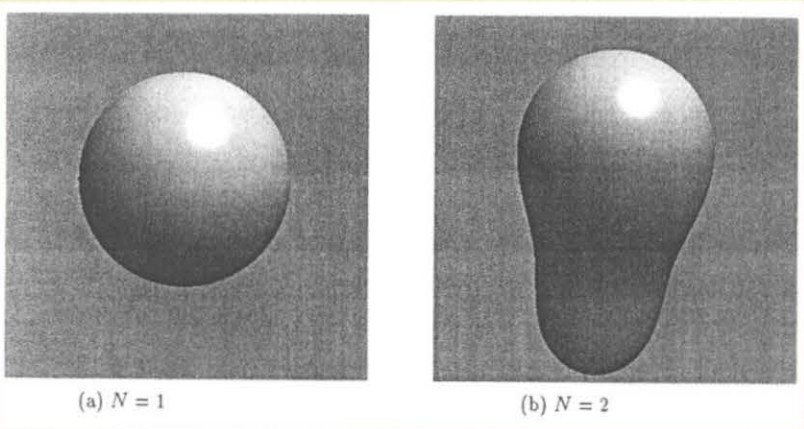
(a) $N = 1$ (b) $N = 2$



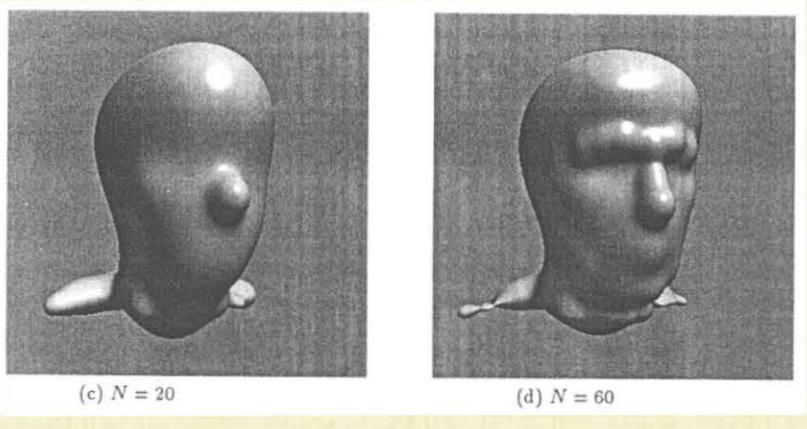
(c) $N = 10$ (d) $N = 35$



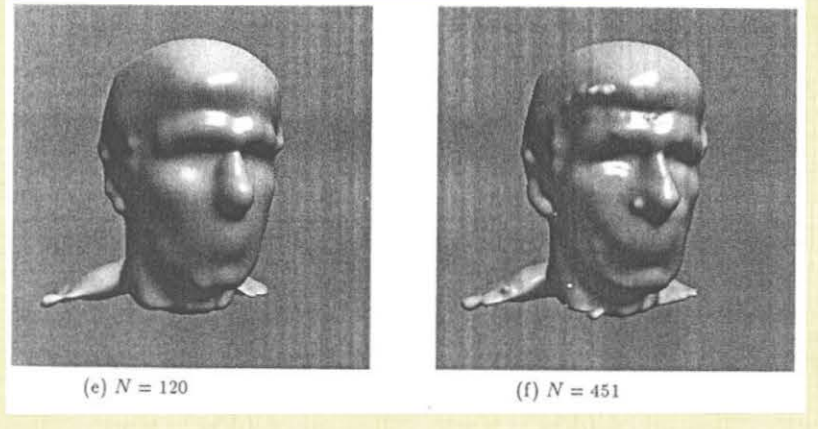
(e) $N = 70$ (f) $N = 243$



(a) $N = 1$ (b) $N = 2$



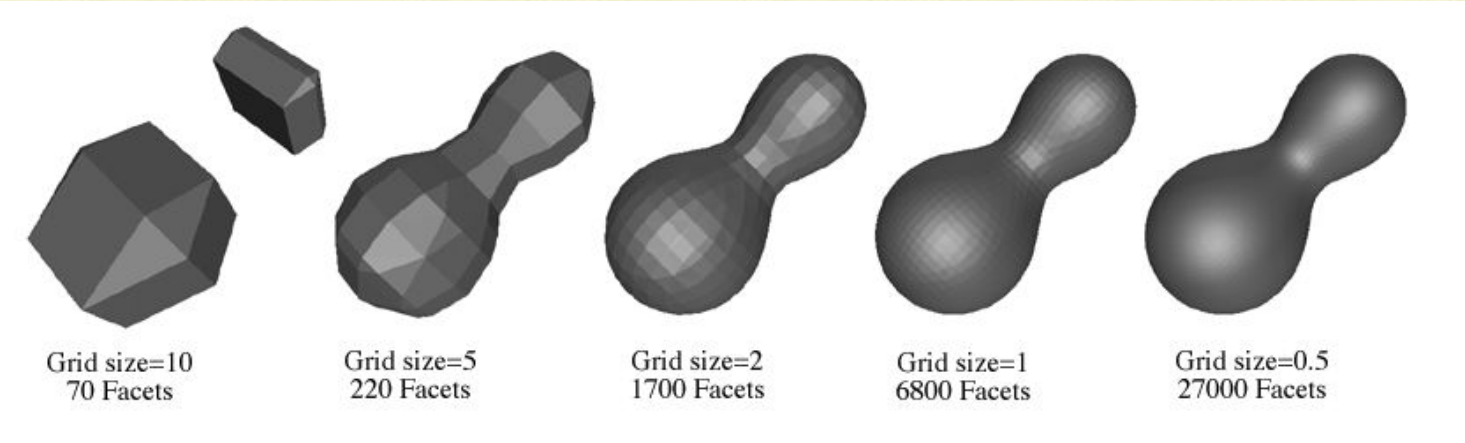
(c) $N = 20$ (d) $N = 60$

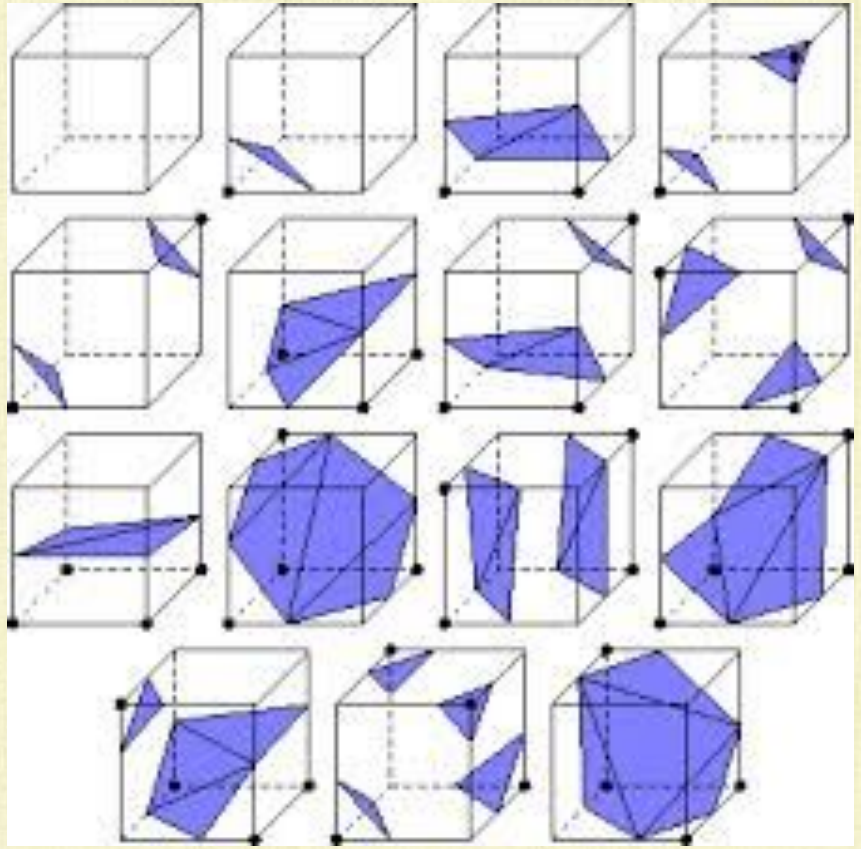


(e) $N = 120$ (f) $N = 451$

Marching Cubes (or Tetrahedra)

- Turns an implicit surface into triangles
- Define the implicit surface on a 3D grid
- Then for each grid cell, use the topology of the volume in order to reconstruct the surface with triangles

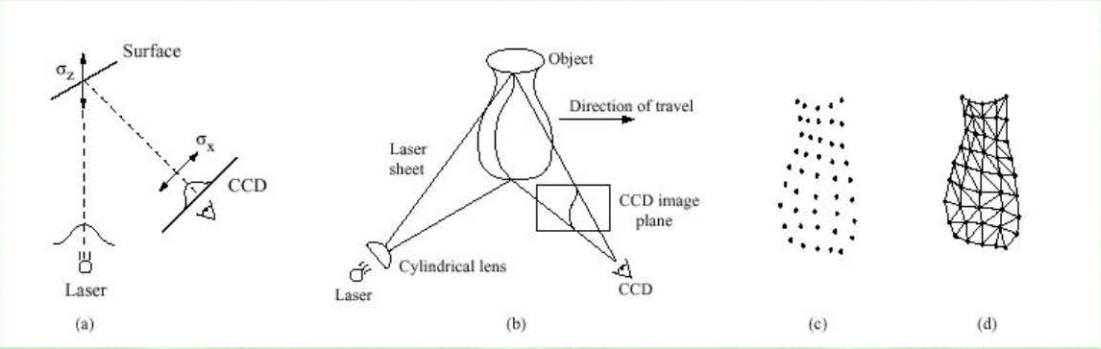


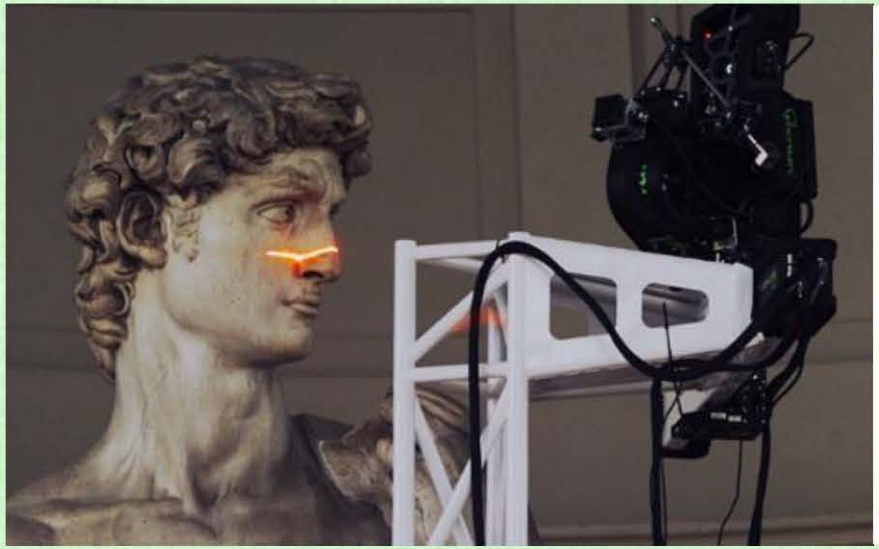


Computer Vision

Range Scanning

- A range scanner senses 3D positions on an object's surface and returns an $m \times n$ grid of distances that describe the surface (m points per laser sheet, n laser sheets)
  - This grid is called a range image
  - In case of multiple range images, a rigid transformation is found (for each image) to align them together
  - This transformation is found using an Iterative Closest Point alignment (ICP), which minimizes the least squared distance between nearest points in two range images





Range Scanning

- Each sample point in the $m \times n$ range image is a potential vertex in the triangle mesh
- Special care is required to avoid inadvertently joining portions of the surface together that are separated by depth discontinuities
- The meshes corresponding to each range image are combined using the zippering algorithm given the rigid transformation between each image

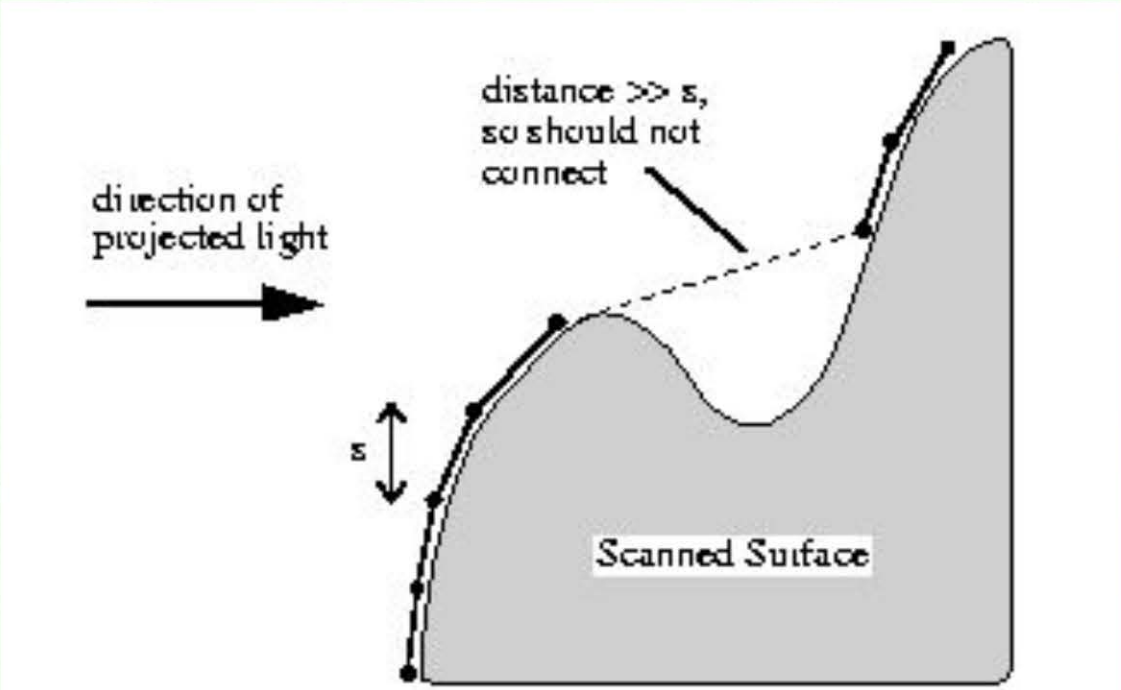
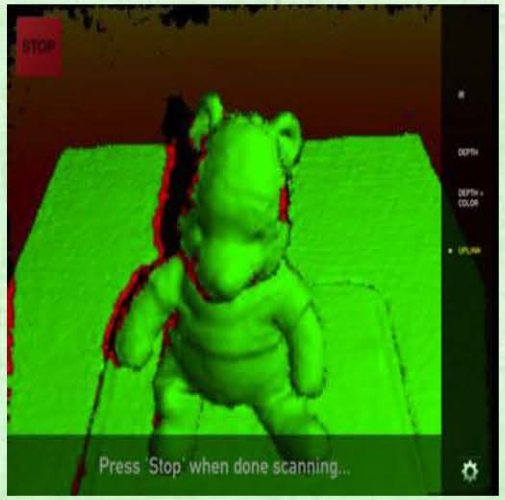


Figure 3: Building triangle mesh from range points.

Mobile 3D Scanning



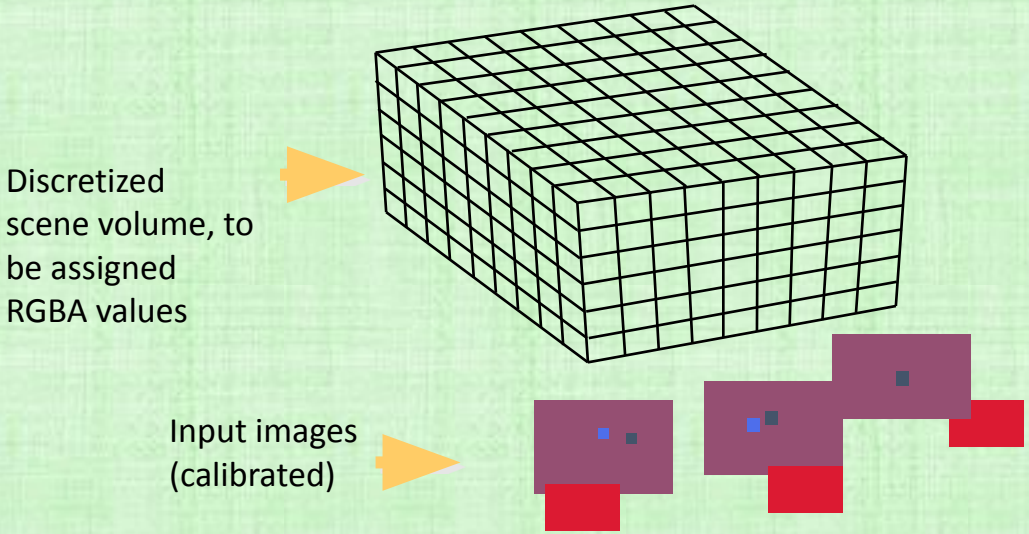
Structure Sensor for iPad

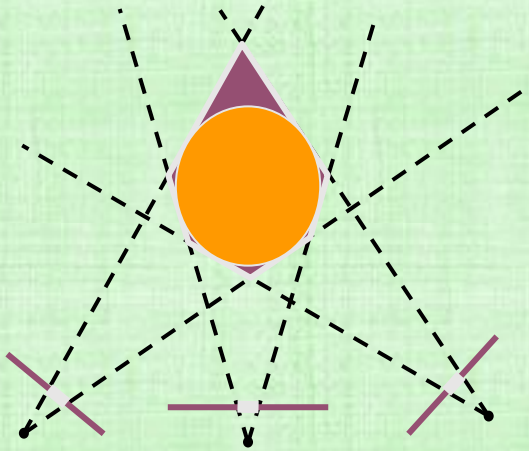
AUTODESK® 123D®

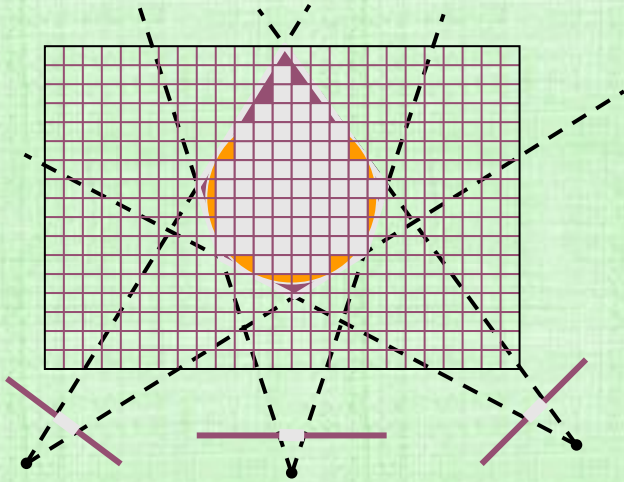
Autodesk 123D Catch

Voxel Carving

- Construct a voxelized 3D model given multiple images of an object (from calibrated cameras), taken from different directions
  - A silhouette is computed for each image
  - The silhouette (for each image) is back projected onto the grid
  - The voxels that lie in the back-projection of every image correspond to the final 3D model
  - Once the geometry is acquired, colors can also be back projected



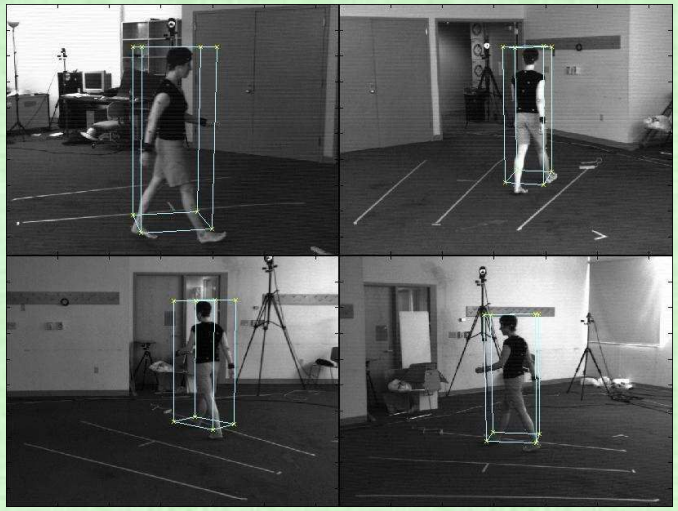




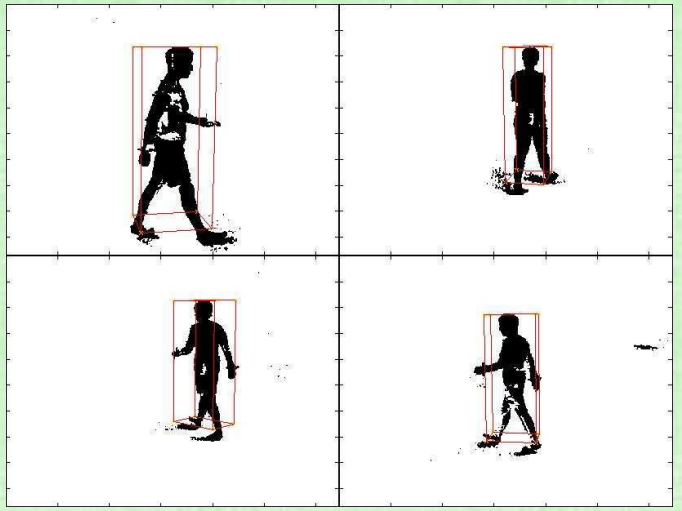
Color the voxel gray if silhouette is in every image

Voxel Carving

Original image

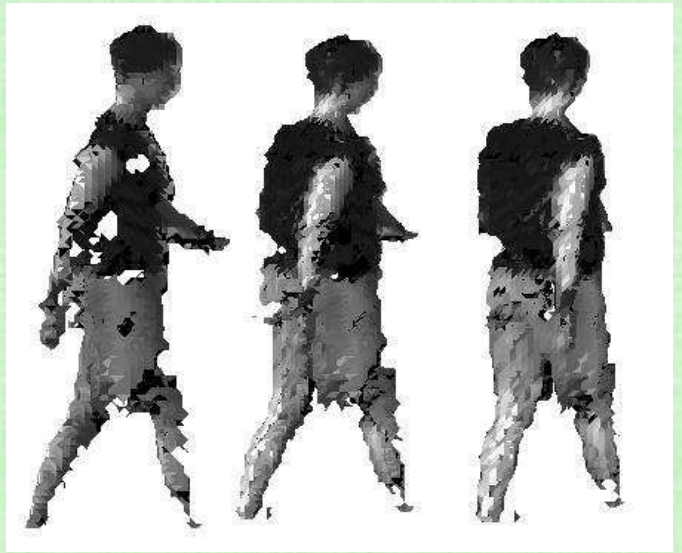


Extracted silhouettes



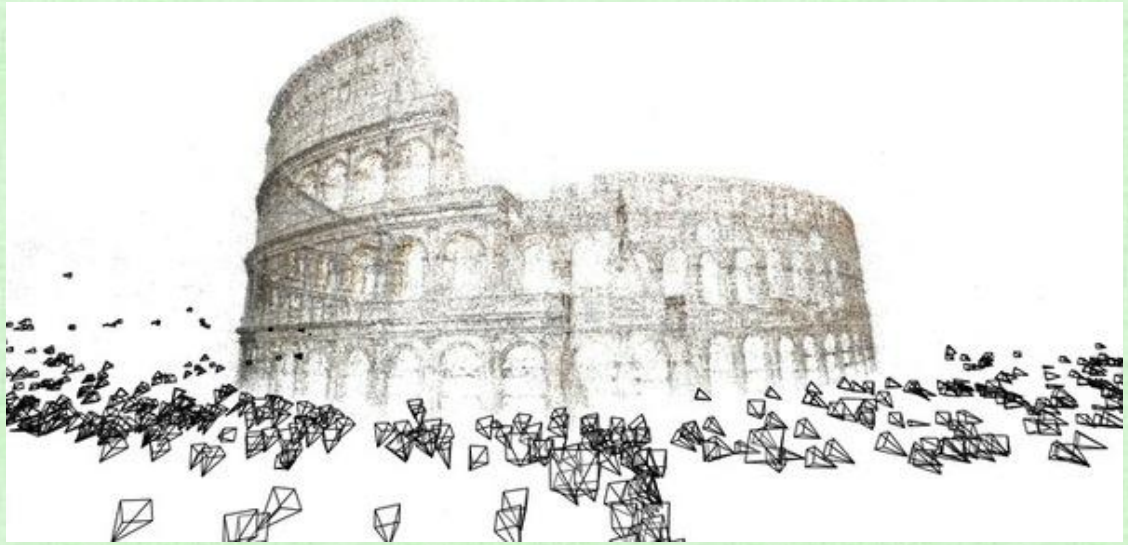
Carved out voxels

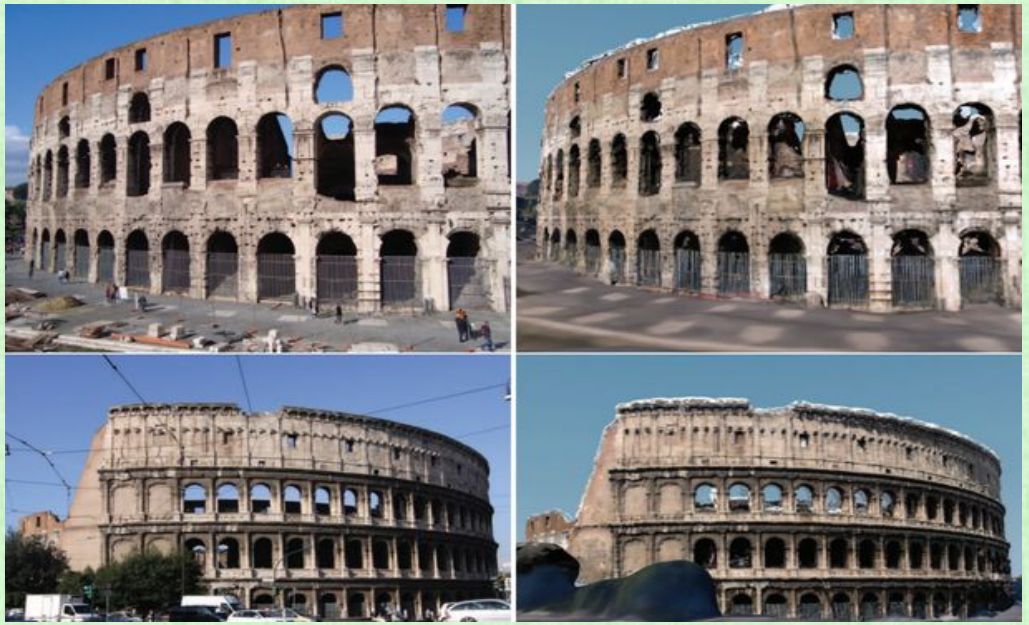
Back-projecting colors



Reconstruction from Large Photo Collections

- Construct a 3D model from a large number of photos (say from google images)
- Computer vision algorithms are used to predict relative camera position/orientation for each image, and to obtain a sparse point cloud representation of the object
- The position of a point that is visible in multiple images can be determined
- Many dense reconstruction algorithms can be used to get denser points given the camera parameters and this initial point cloud

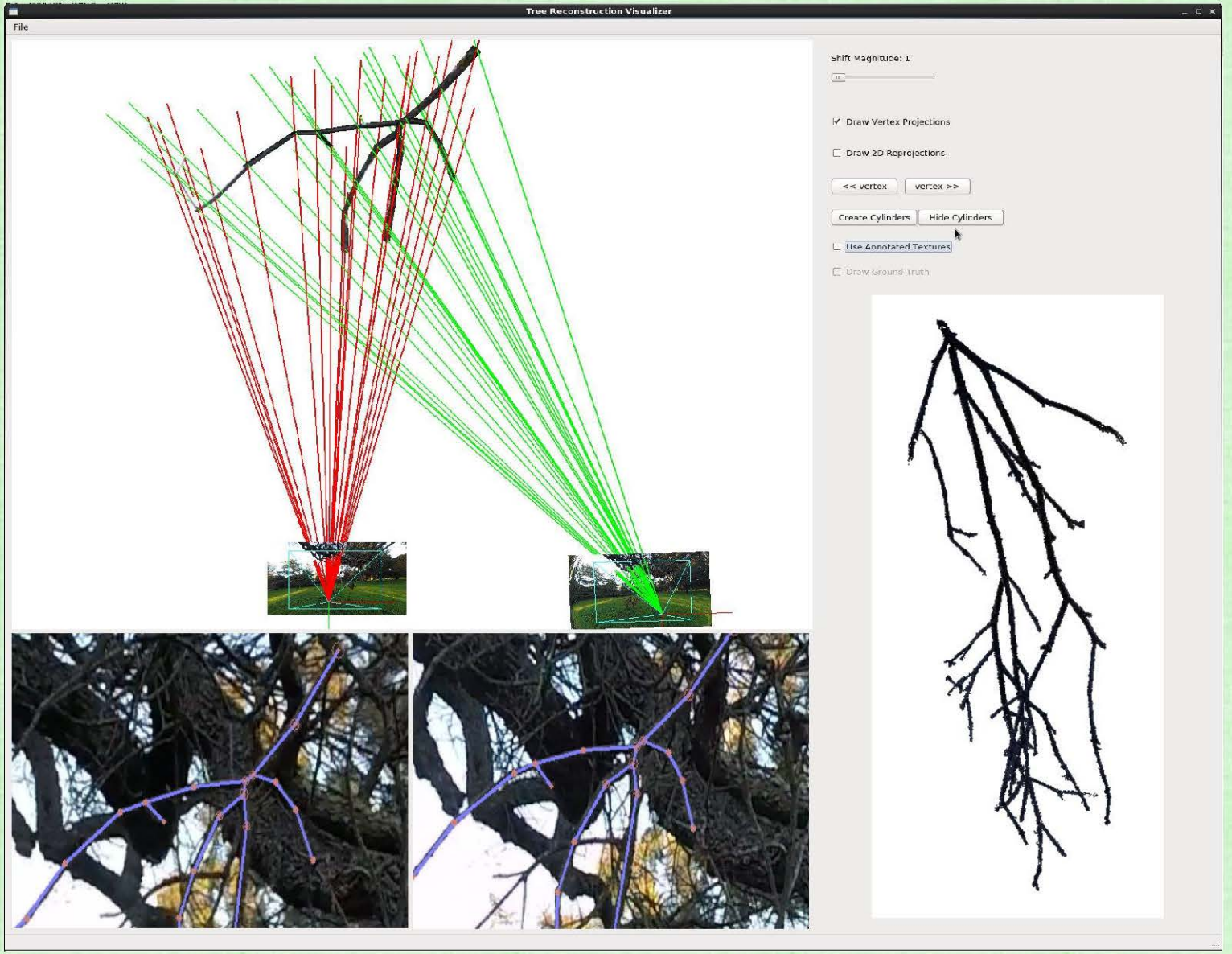




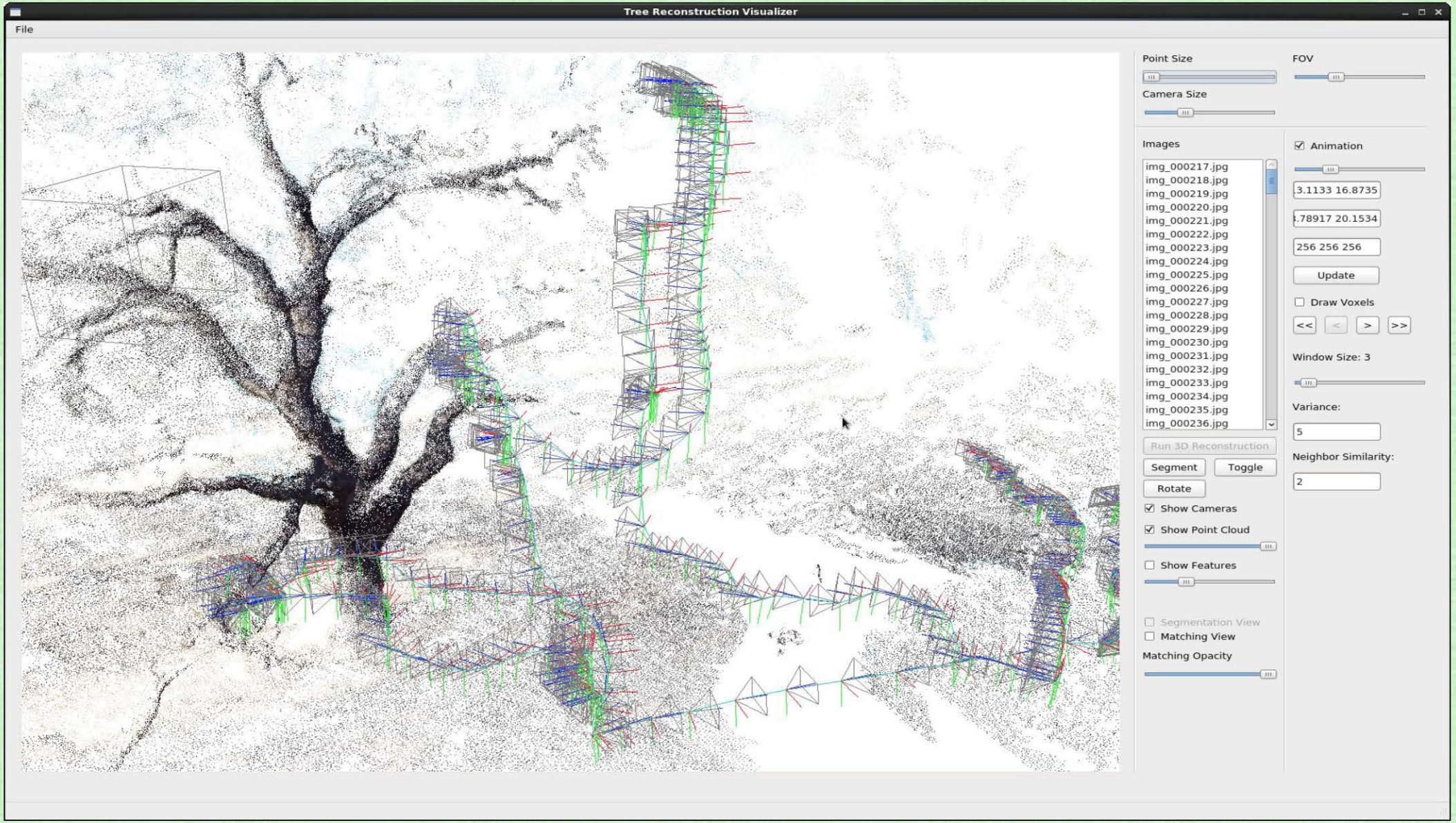
Real Photos

3d Reconstructed models

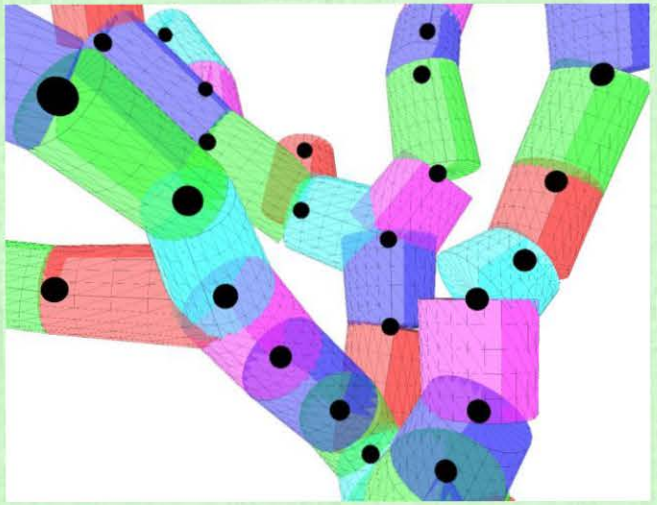
Trees, Drones, Etc.



Trees, Drones, Etc.



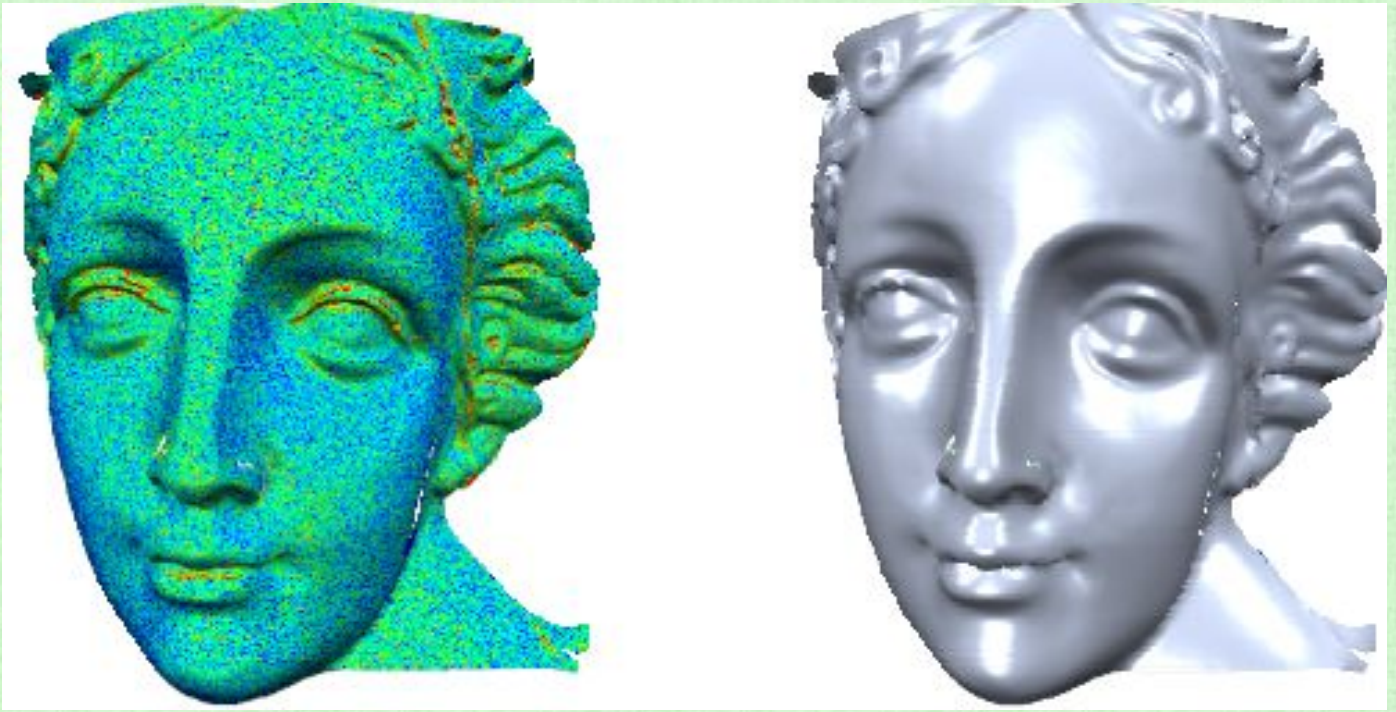
Trees, Drones, Etc.



Trees, Drones, Etc.

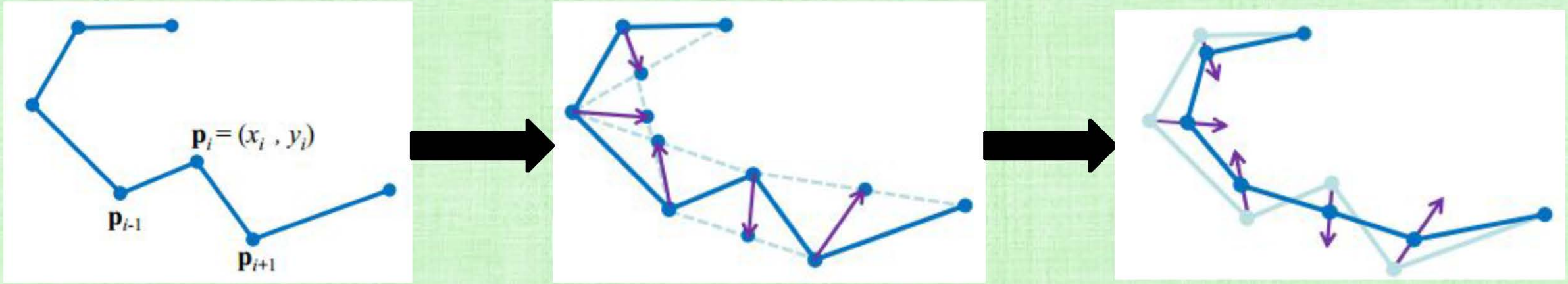
Denoising

- Computer Vision algorithms construct virtual geometry from real world geometry
- This requires collecting data from the real world, and this data collection is highly sensitive to noise
- Noise is probably the biggest drawback to such methods
- Thus, various methods have been devised to denoise/smooth high problematic geometry with spurious high frequency features



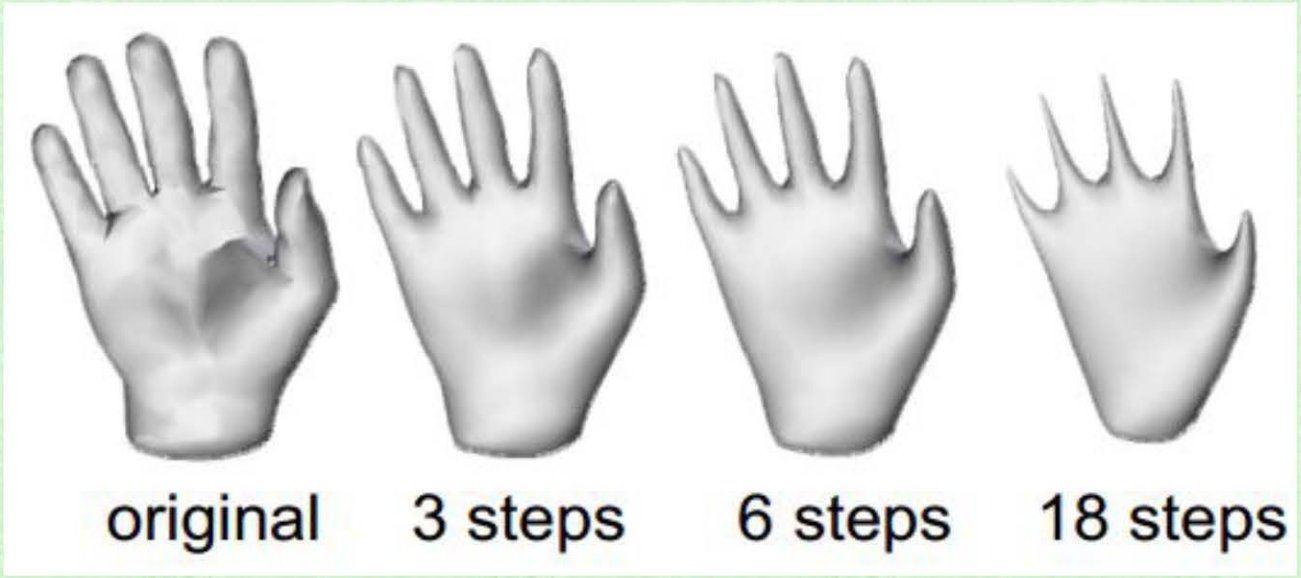
Laplacian Smoothing

- Similar to differential coordinates, one can compute a Laplacian estimate using the one ring of vertices about a point
- For example, on a curve (see below), one might use $L(p_i) = \frac{1}{2}\left((p_{i+1} - p_i) + (p_{i-1} - p_i)\right)$
- Then, update $p_i^{new} = p_i + \lambda L(p_i)$ where $\lambda \in (0,1)$
- Repeat several iterations

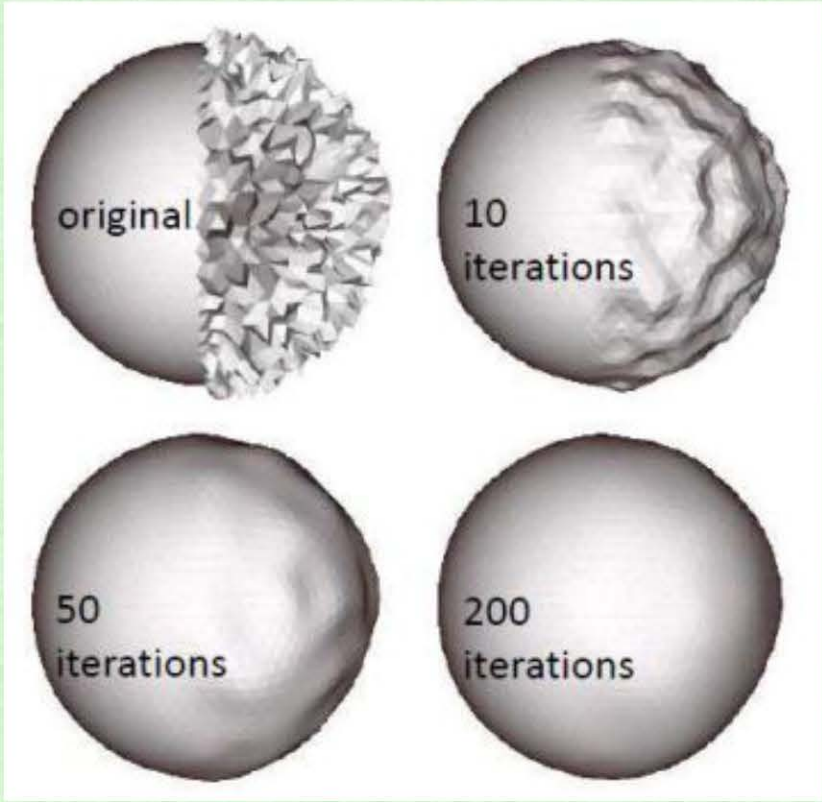


Taubin Smoothing

- Laplacian smoothing eventually shrinks a closed curve/surface to a single point
- Taubin smoothing periodically performs an extra inflation step to counteract the shrinkage due to Laplacian smoothing: $p_i^{new} = p_i - \mu L(p_i)$ where $\mu > 0$



Laplacian smoothing (only)



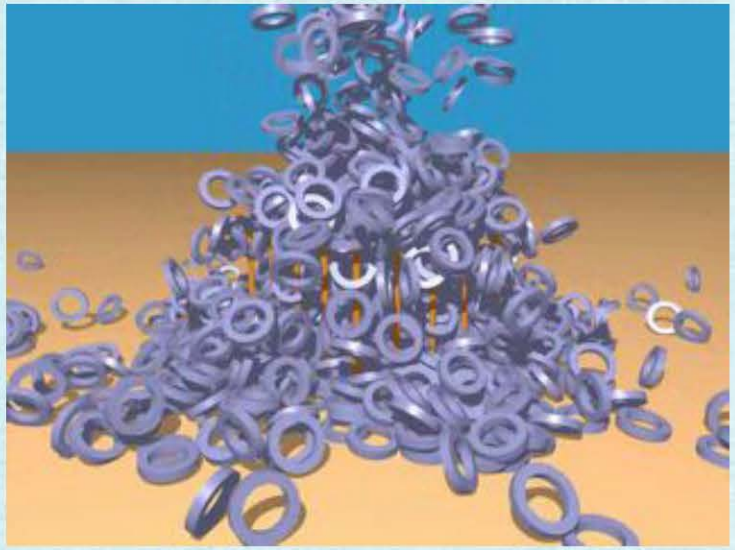
Taubin smoothing

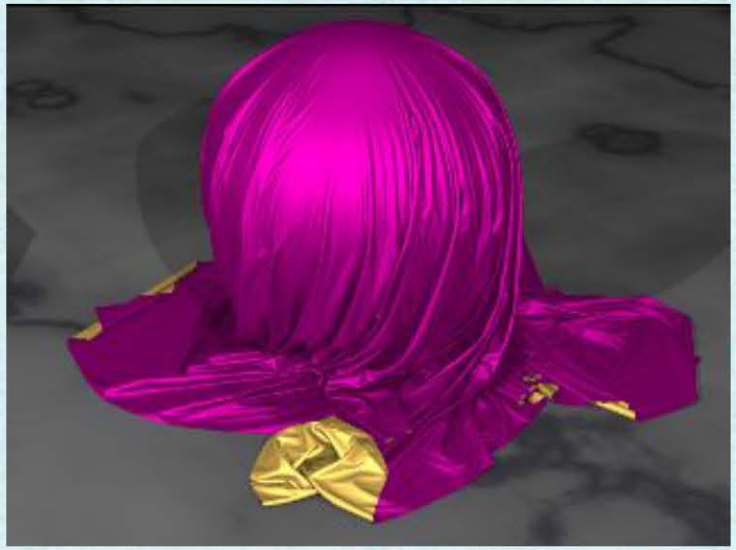
Simulation

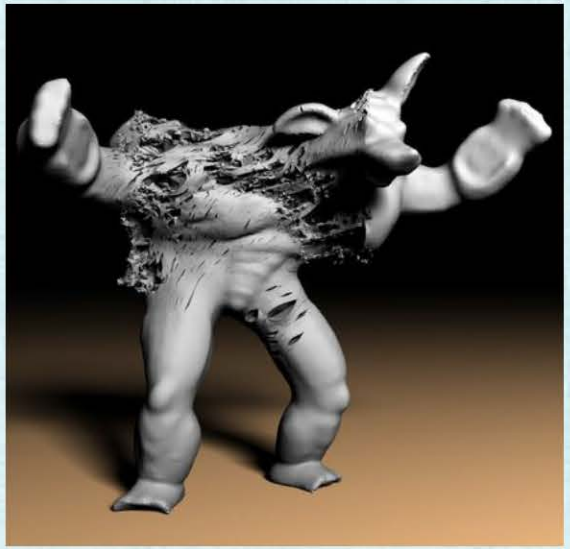
Netwon's Second Law

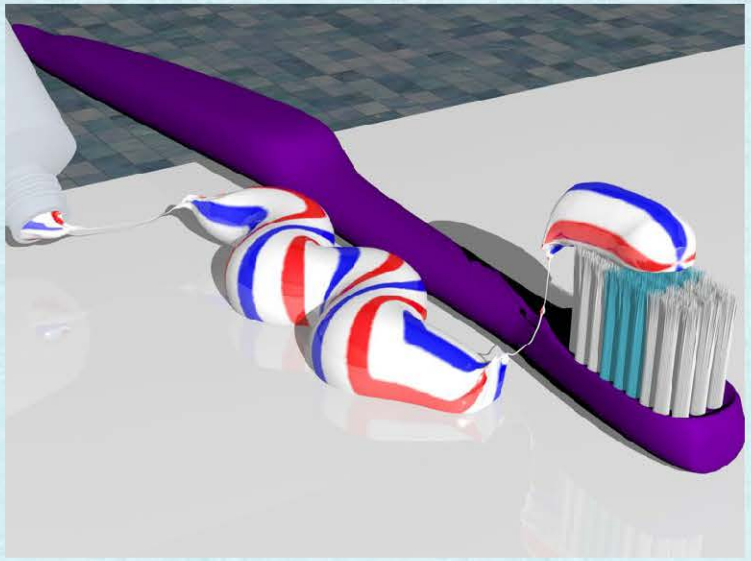
- Kinematics describe position $X(t)$ and velocity $V(t)$ as function of time t
  - $\frac{dX(t)}{dt} = V(t)$ or $X'(t) = V(t)$
- Dynamics describe responses to external stimuli
  - Newton's second law $F(t) = MA(t)$ is a dynamics equation
  - $V'(t) = A(t)$ implies $V'(t) = \frac{F(t)}{M}$ as well as $\frac{\mathrm{d}^2X(t)}{dt^2} = X''(t) = \frac{F(t)}{M}$
- Combining kinematics and dynamics gives: $\begin{pmatrix} X'(t) \\ V'(t) \end{pmatrix} = \begin{pmatrix} V(t) \\ \frac{F(t,X(t),V(t))}{M} \end{pmatrix}$
  - Note the (potential) dependency of forces on position and velocity
- Much of the physical world can be simulated with computational mechanics (FEM) and computational fluid dynamics (CFD) using Newton's second law
  - One merely needs to create degrees of freedom, specify forces, and solve the resulting ordinary differential equations (ODEs)
  - For most materials, the forces have spatial interdependence making this a partial differential equation (PDE)

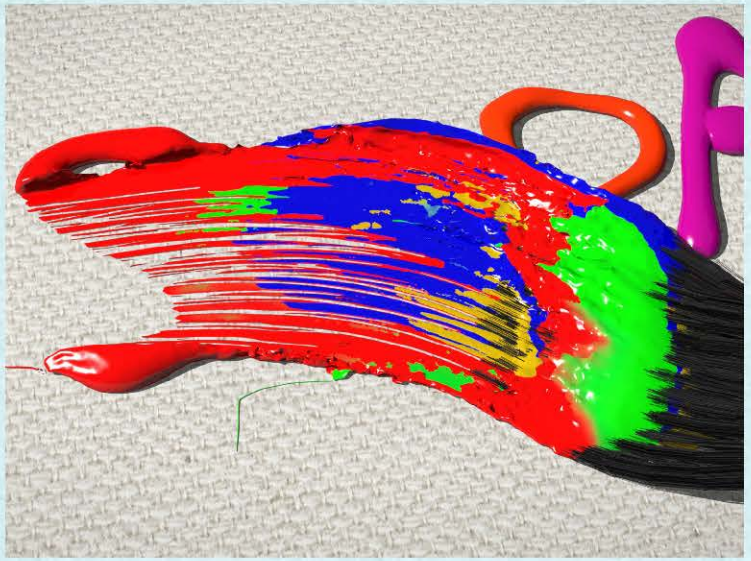
Computational Mechanics (FEM)

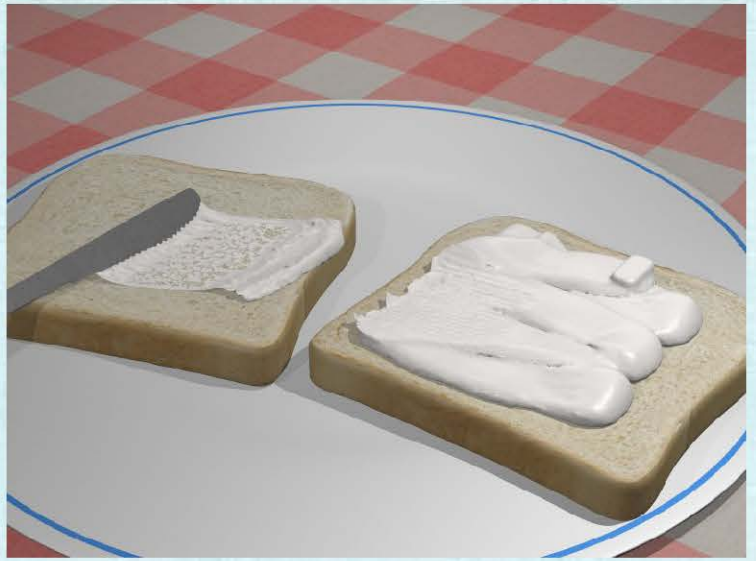




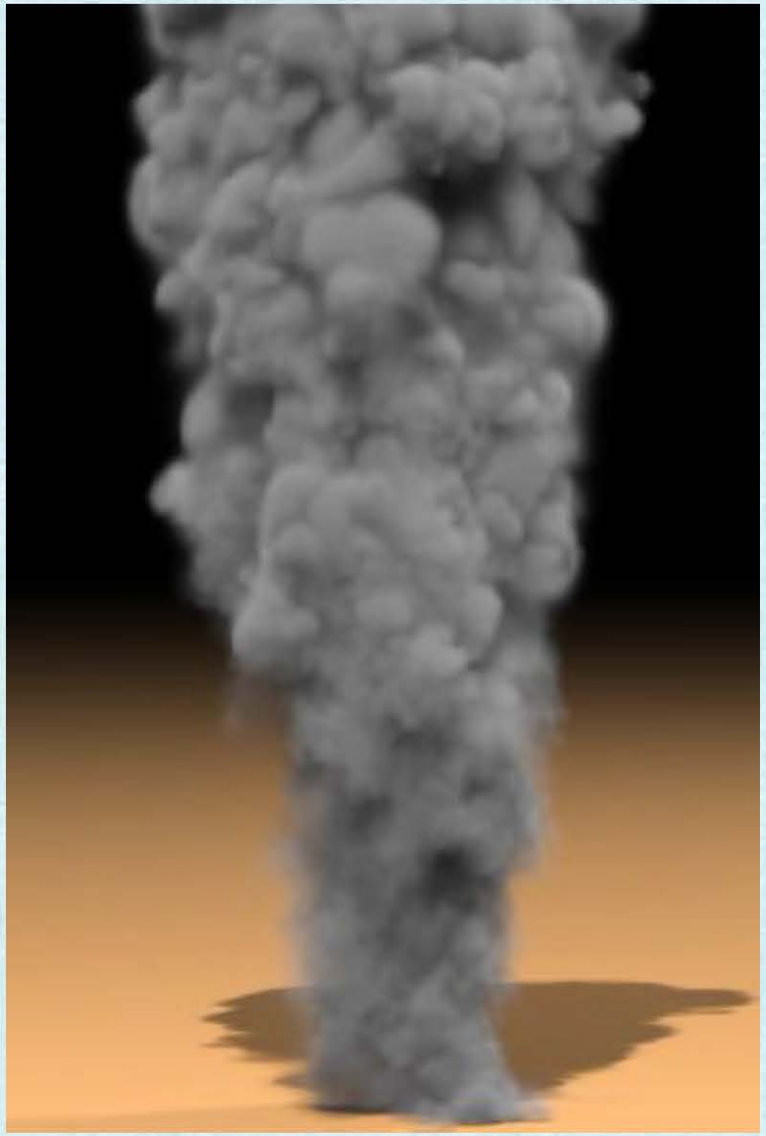


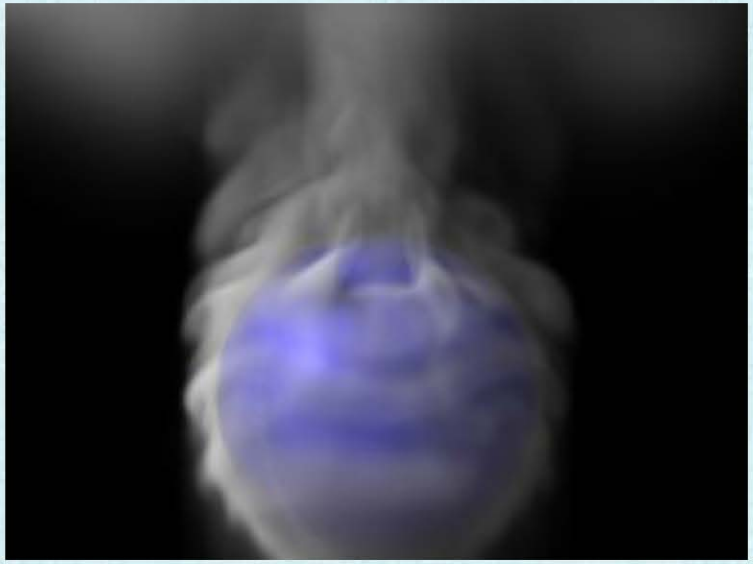


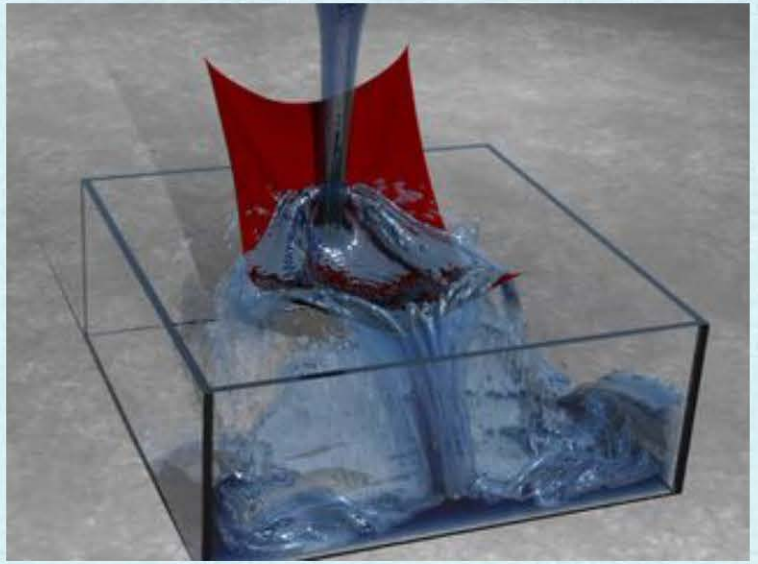




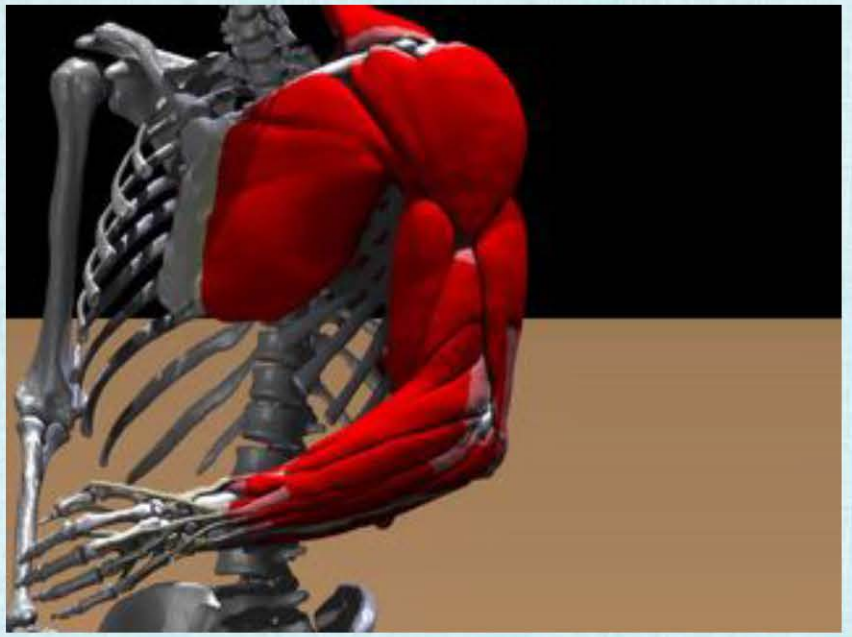
Computational Fluid Dynamics (CFD)

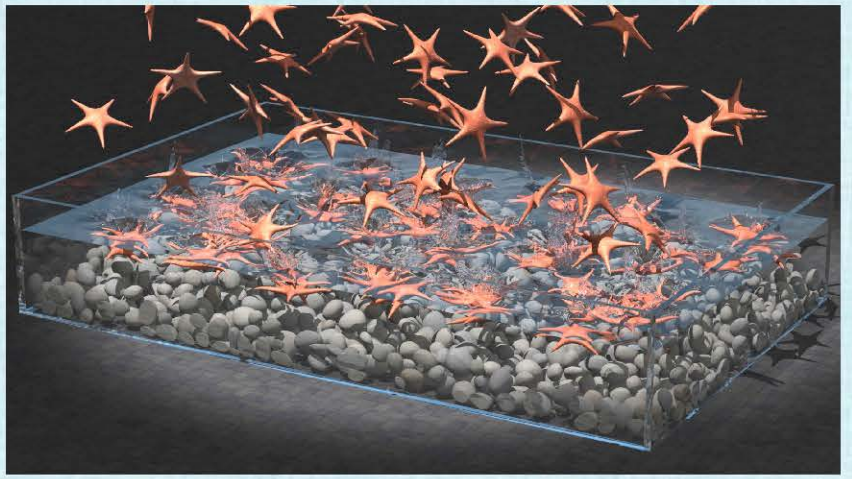






Computational Biomechanics

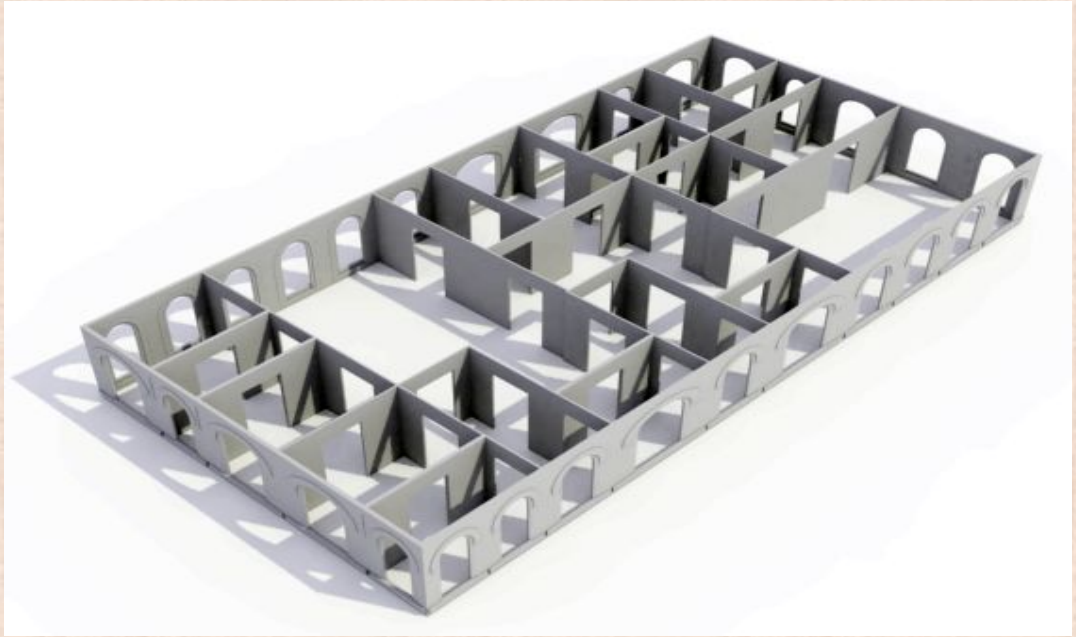


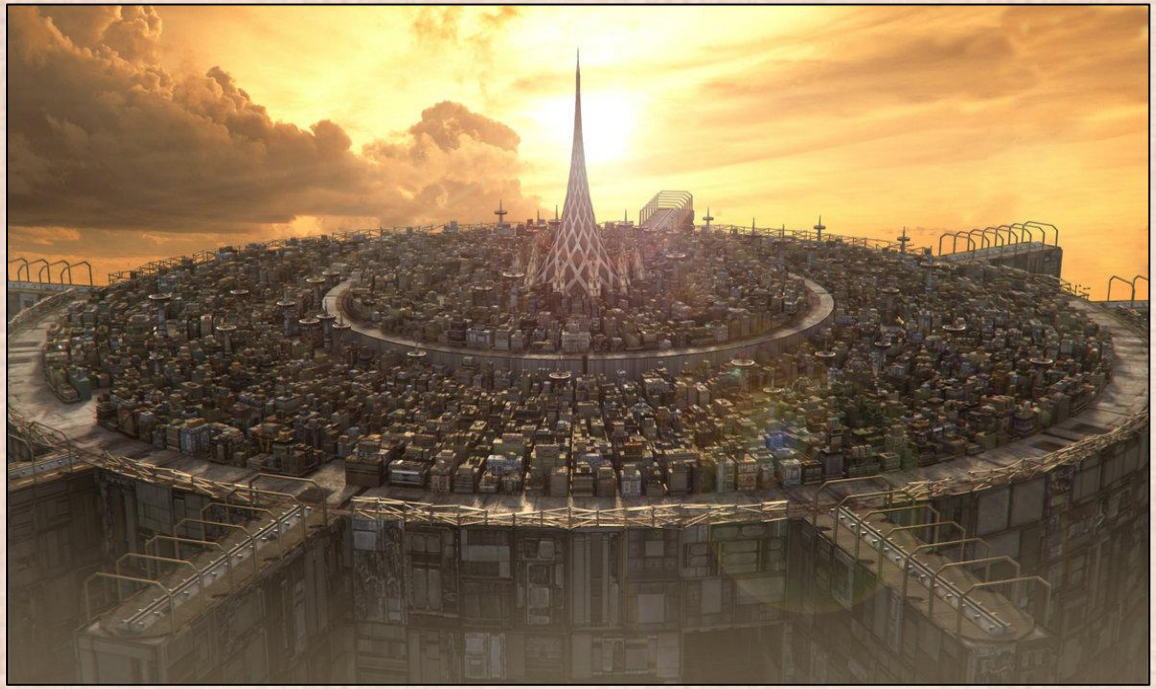


Procedural Methods

Procedural Geometry

- Whereas simulation uses algorithms to animate existing geometry, one can also use an algorithm to create the geometry itself
  - Typically used for complex/tedious models (e.g. terrain, plants/foliage, buildings/cities)
  - Easy to perturb the algorithm to make variations of the geometry
- Start with a small set of data or rules to describe high level properties of the desired models
  - E.g. tree: branching and leaf shape, building: room subdivision and door/window placement
- The rest of the model is algorithmically constructed (add randomness and use recursion)





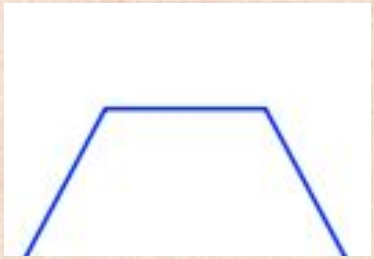
L-Systems

- Typically used to model plants
  - Developed by biologist Lindenmayer to study algae growth
- A recursive formal grammar:
  - An alphabet of symbols (terminal and non-terminal)
  - A collection of production rules
  - Non-terminal symbols create new symbols or sequences of symbols recursively
- The process starts with an initial string (axiom) to which the production rules are applied
- Finally, a translator turns both terminals and non-terminals into geometric structures

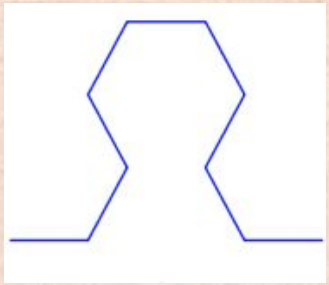
Nonterminals: A, B: mean “draw forward”
Terminals: +/- : Turn right/left by 60 degrees
Initial Axiom: A; Rules: A → B + A + B, B → A − B −

A

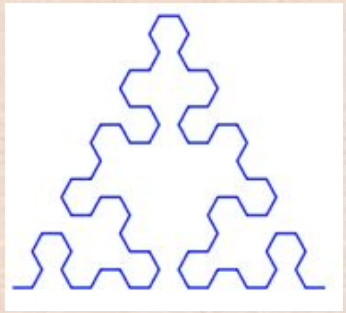
A



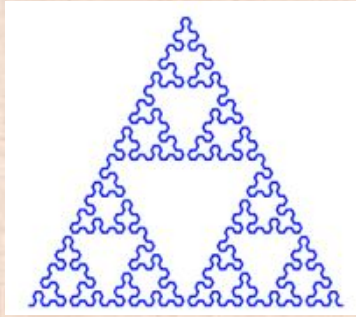
B+A+B



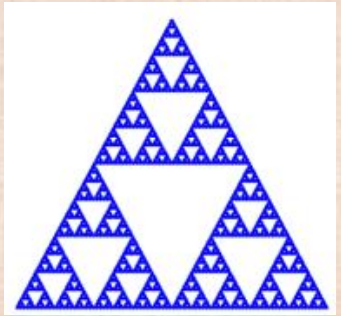
A-B-A + B+A+B + A-B-A



B+A+B - A-B-A - B+A+B
+ A-B-A + B+A+B + A-B-A
+ B+A+B - A-B-A - B+A+B



Etc.



Sierpinski Triangle

L-System + Stack = Branches

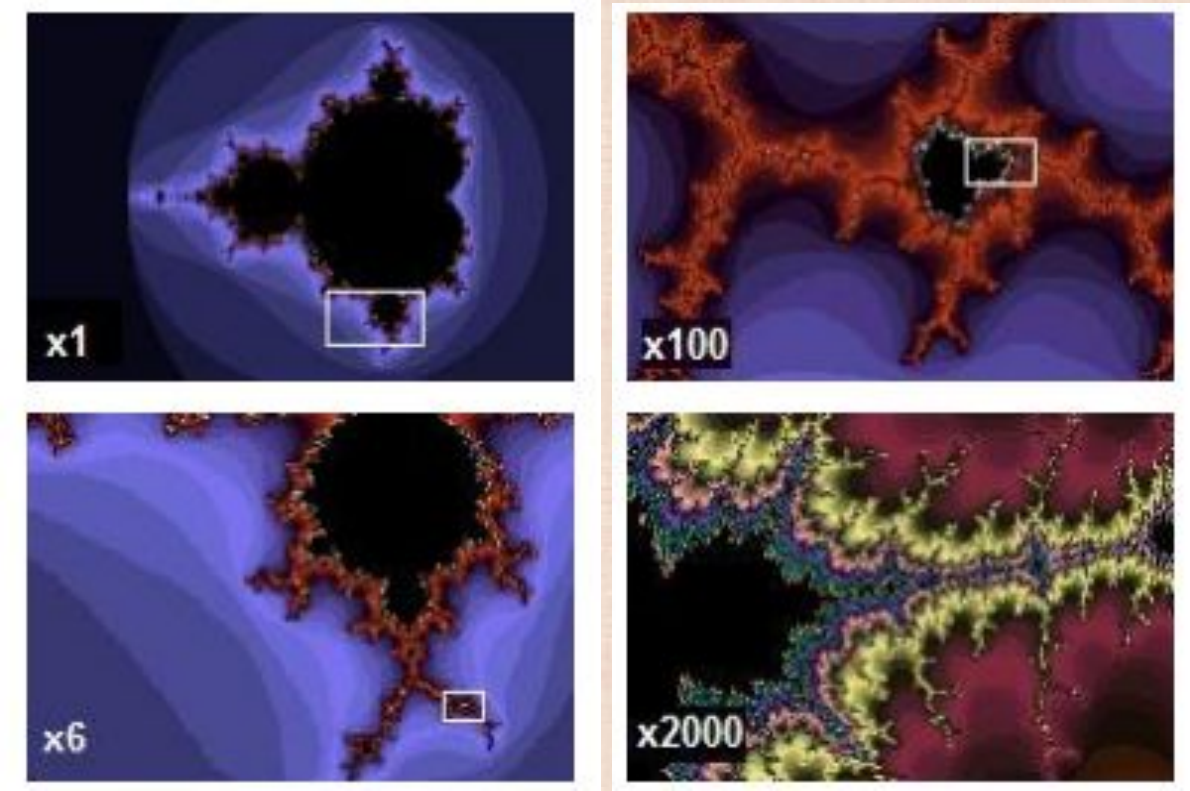
- Nonterminals: X: (no action) F: draw forward
- Terminals: +/- : Turn right/left by 25 degrees
  - **[: store current state on the stack**
  - **] : load state from stack**
- Initial Axiom: X
- Rules: X → F − [[X]+X] + F [+FX] − X, F → FF

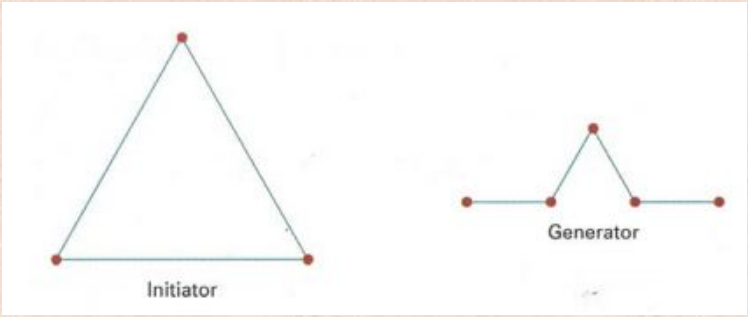
L-Systems

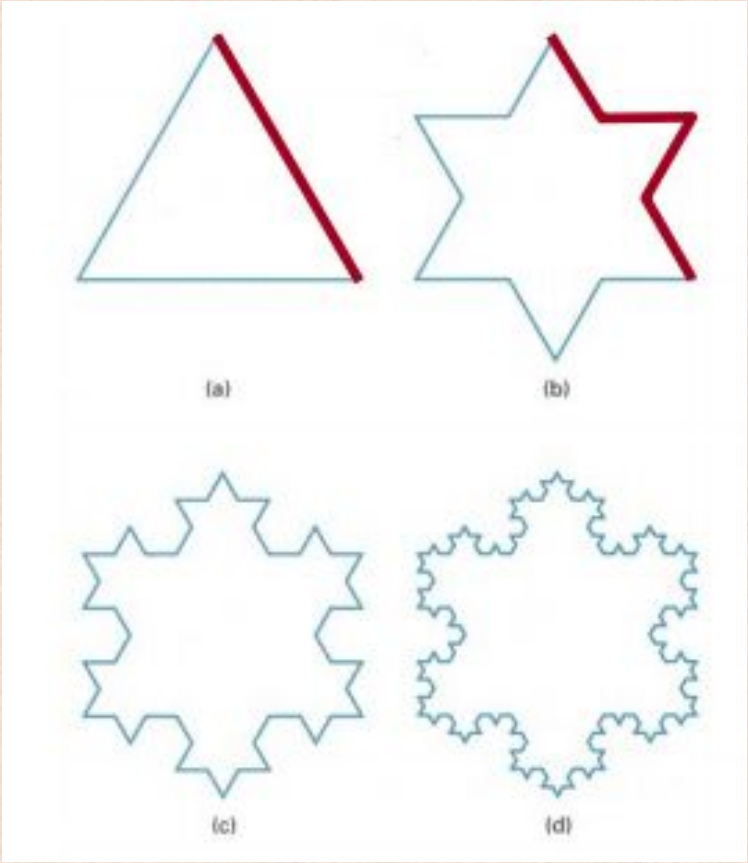
- Easily extended to 3D
  - Model the trunk and branches as cylinders
  - As recursion proceeds:
    - Shrink cylinder size
    - Vary color from brown to green
  - Add more variety with a stochastic L-system
    - Multiple rules for each symbol
    - At each symbol, randomly choose one rule to replace
  - L-system is a relatively abstract specification
    - Requires experience to model a specific and given form

Fractals

- Initiator: start with a shape
- Generator: replace subparts with scaled copy of original
- Apply generator repeatedly



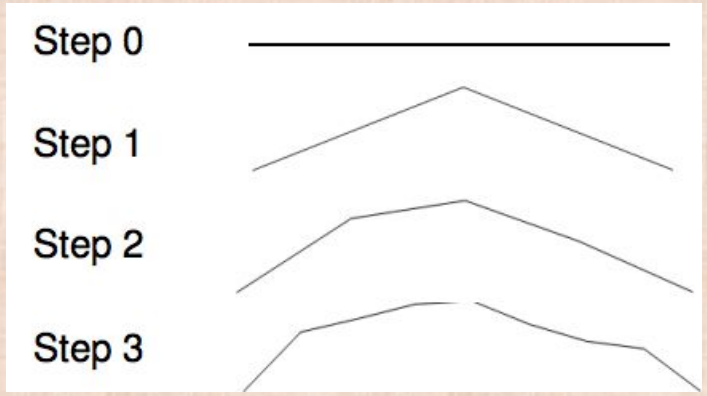


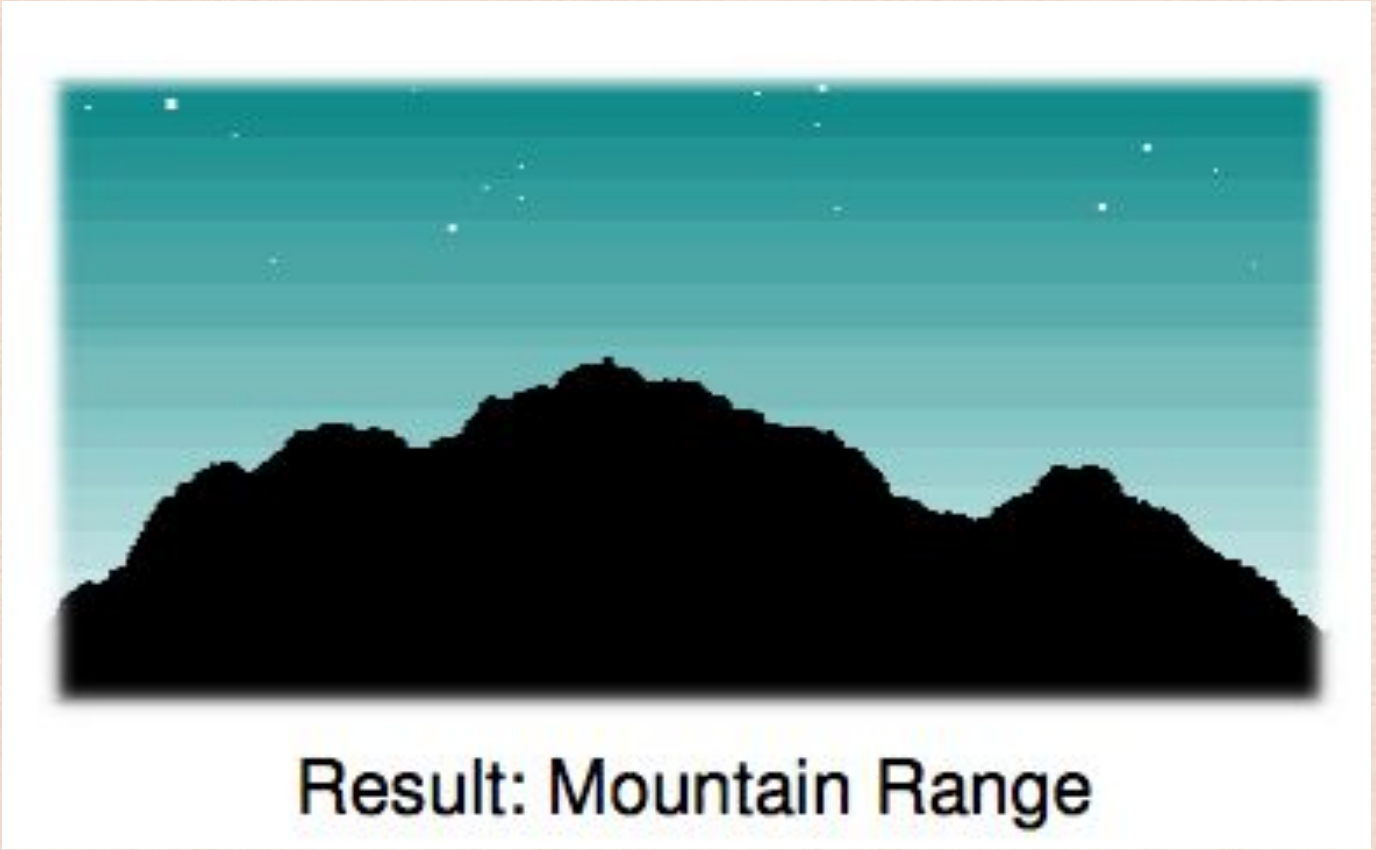


Statistical Fractal Generator

- Add randomness in fractal generation
- Can be used to model an irregular “random” 2D silhouette or terrain
- Random midpoint displacement

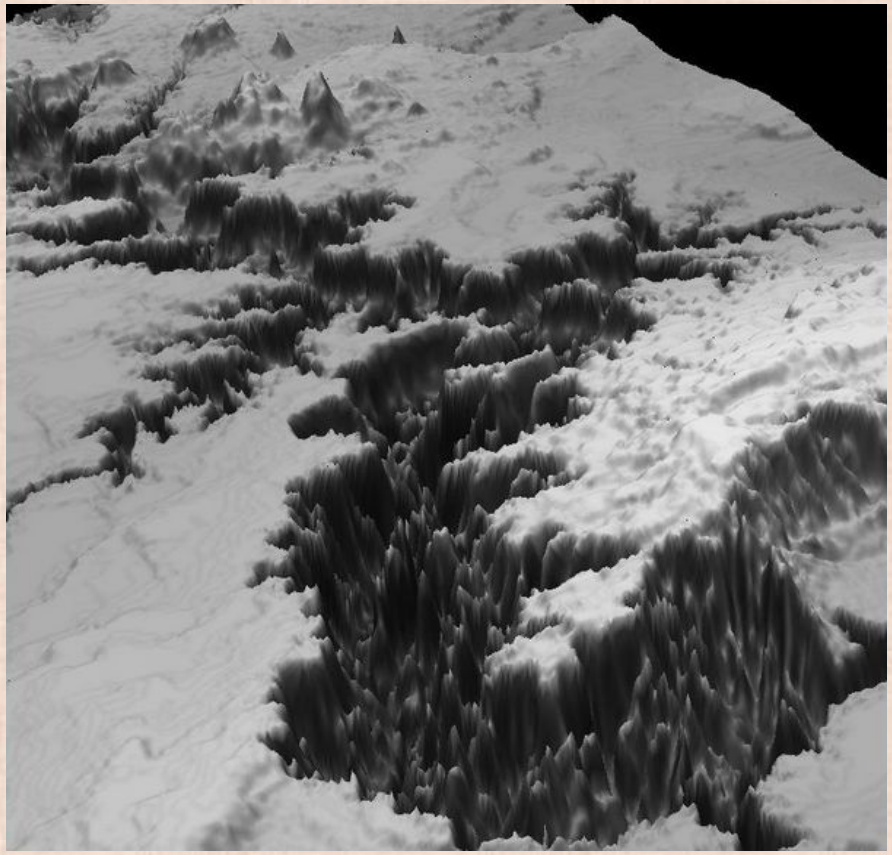
```
Start with single horizontal line segment.
Repeat for sufficiently large number of times
{
  Repeat over each line segment in scene
  {
    Find midpoint of line segment.
    Displace midpoint in Y by random amount.
    Reduce range for random numbers.
  }
}
```





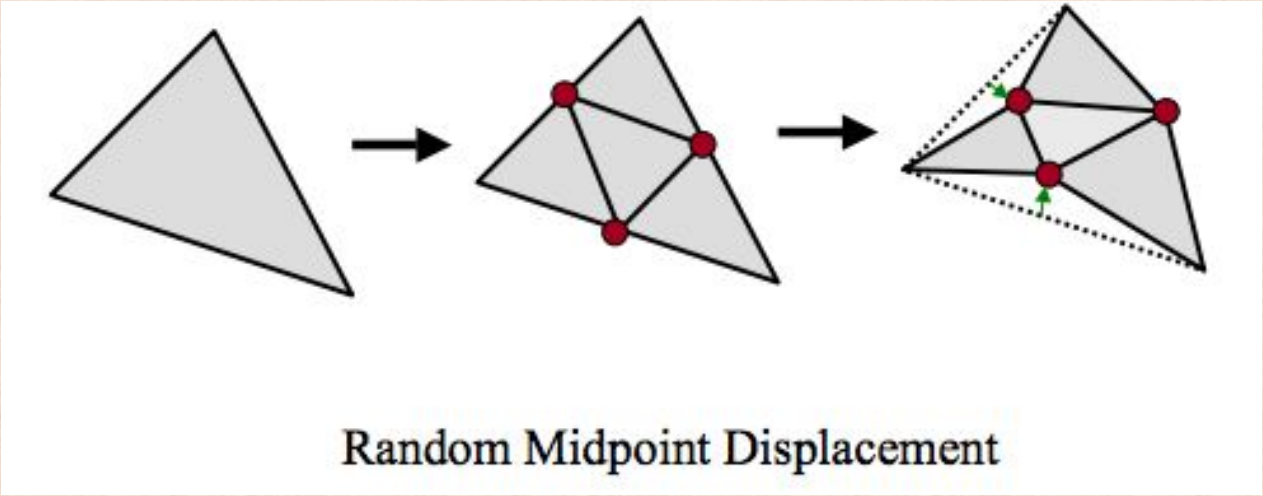
Generating Height Fields

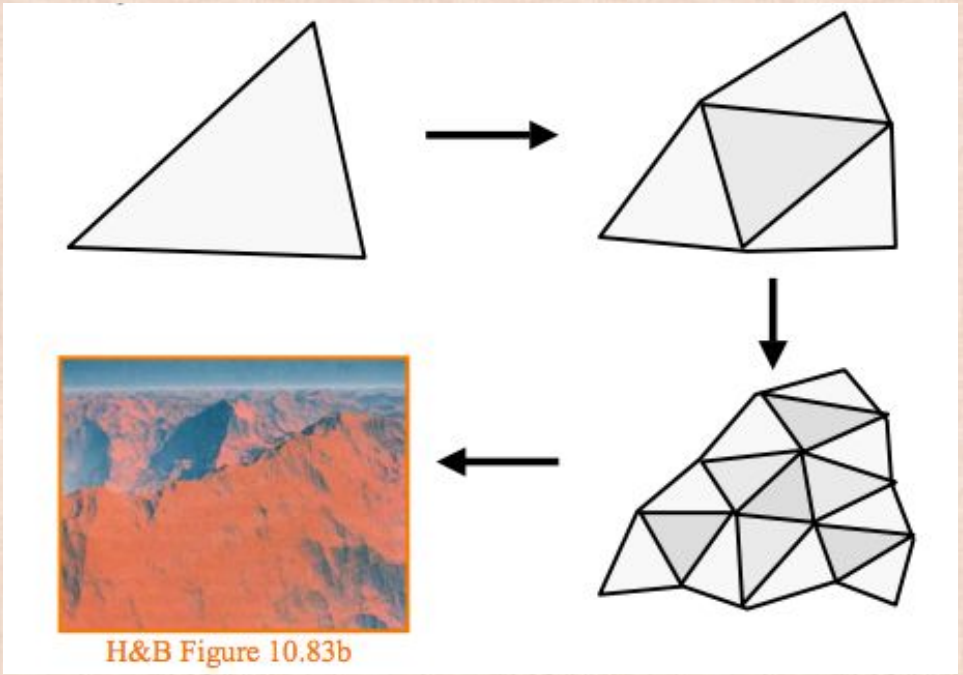
- Start with a 2D fractal (or any 2D grey-scale image)
- Lay a rectangular grid on the ground
- For each vertex of the grid, vary the height based on pixel intensity



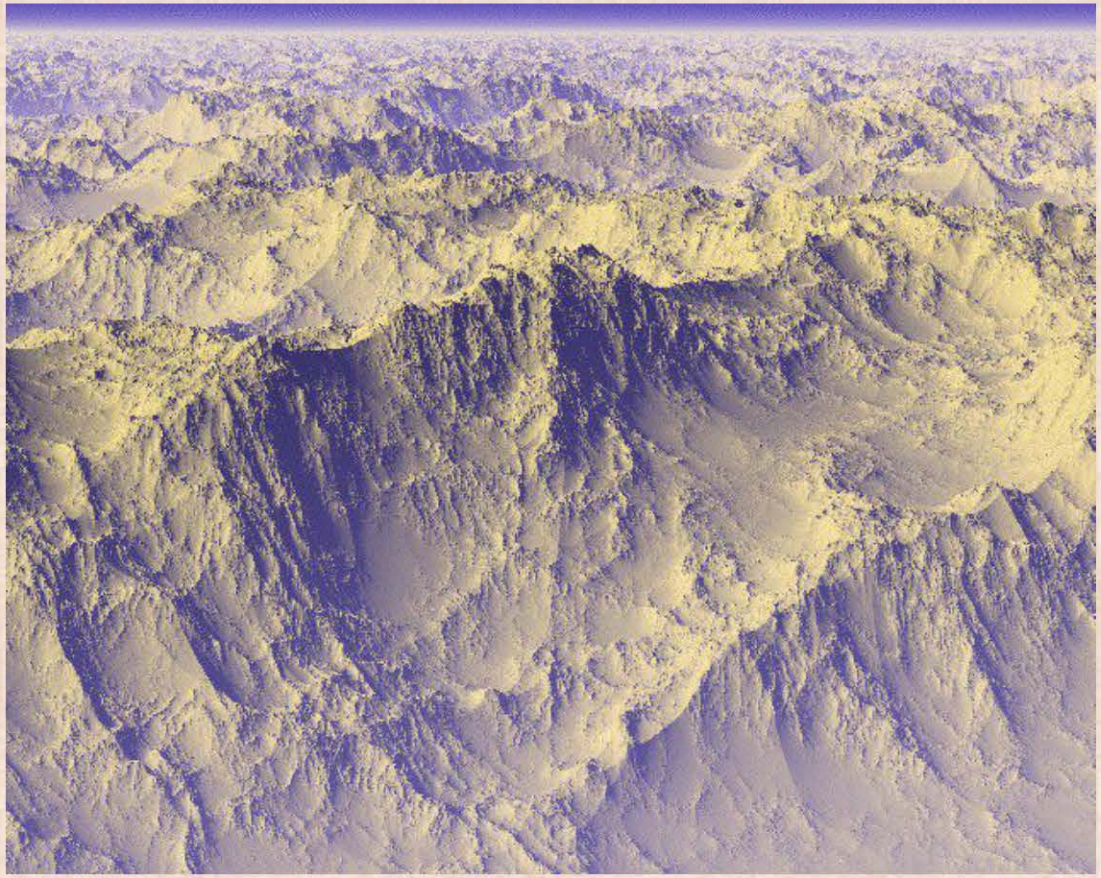
Generating 3D Landscapes

- General procedure:
  - Initiator: start with a shape
  - Generator: random subparts with a self-similar random pattern
- Use this to generate entire terrains
- Similar to subdivision, but with much more interesting rules for setting vertex positions

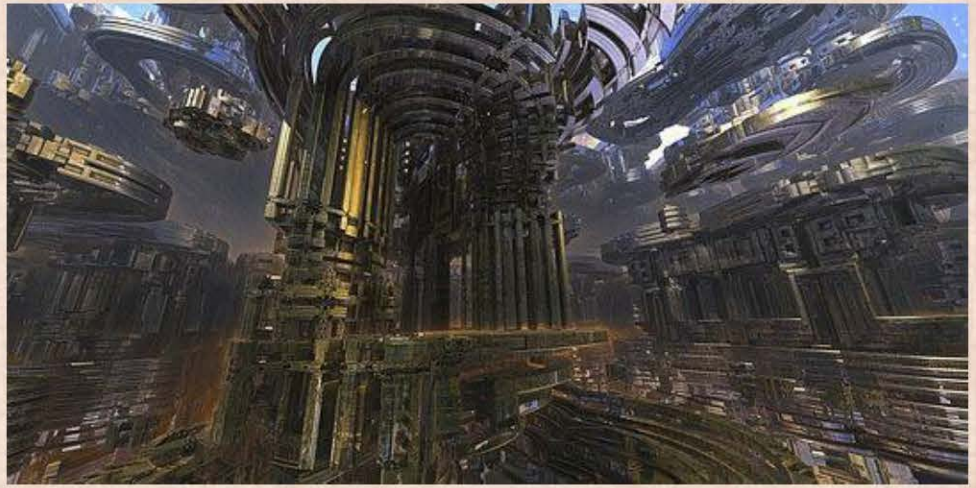


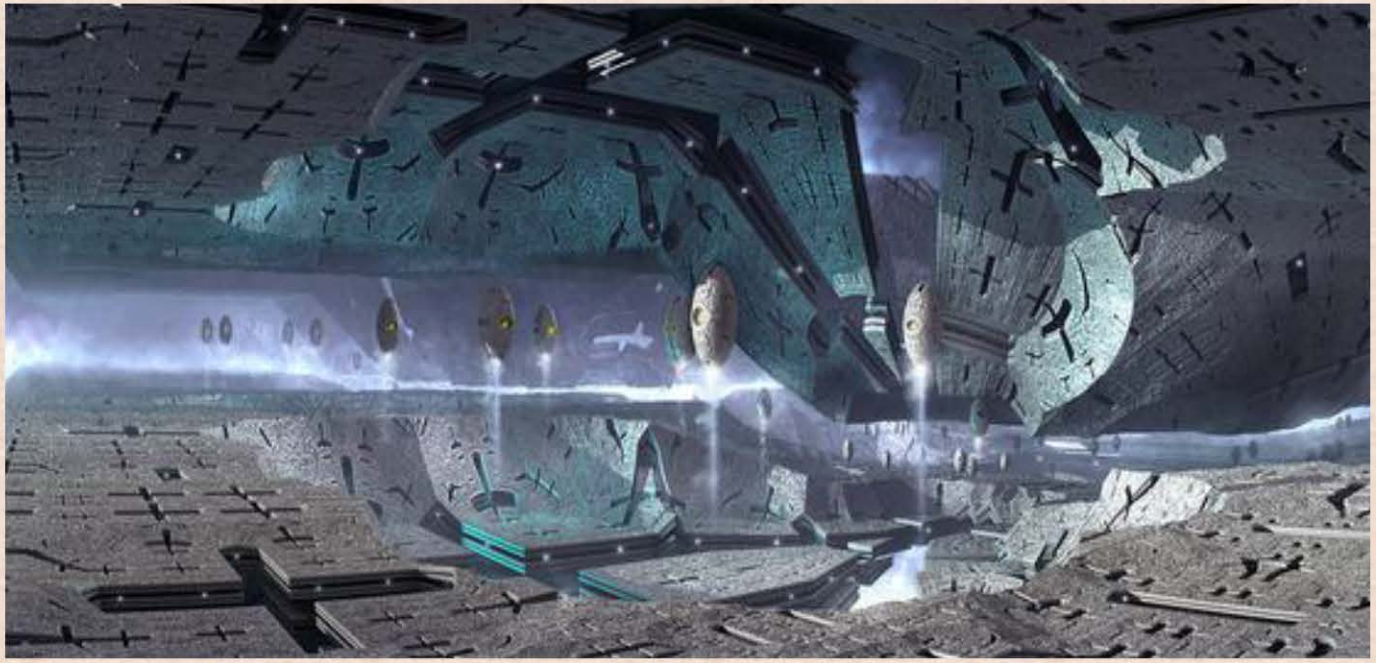


Fractal Worlds



Fractal Worlds





Machine Learning

Machine Learning

- Interactive Example-Based Terrain Authoring with Conditional Generative Adversarial Networks, Siggraph Asia 2017