Modeling Overview

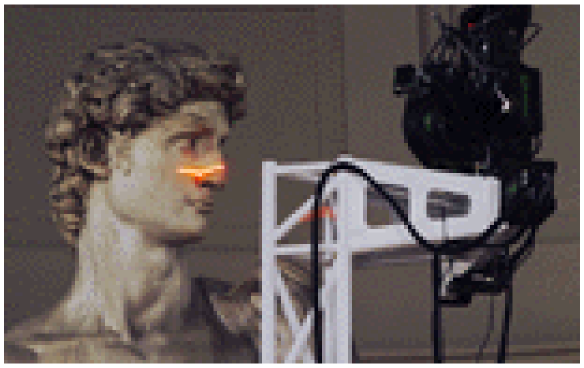
- Goal: Represent 3D objects efficiently allowing for their easy design and modification
- Modeling versus rendering primitives
- Implicit, parametric, procedural modeling

Getting Models

- By hand
- By program
- From the world

3D Sensing with Laser

- Project laser on object
- Triangulate
  - Can be done with known laser position .
  - Or two cameras.
- Project stripe and triangulate on all at once.



Slide 3 Lecture 20

•Triangulation: Given a known direction of light, there is a line in the world where the light might strike the object. This line projects to a line in the image. Location of the point in the image tells us where along this line the light struck.

•Epipolar constraint: Line and focal point form a plane. If we move the light up, we can get a different plane which doesn't intersect this, so the new light direction doesn't interfere.

Slide 4 Lecture 20

Digital Michaelangelo

Slide 5 Lecture 20

Many Other 3D Sensors

- Time of flight sensors
- Stereo
- Medical imaging (eg., MRI, sonar)
- Mechanical

Slide 6 Lecture 20

3D Representations: Triangles

- Excellent rendering primitive:
  - Edges are straight (linear)
  - Interior is flat: incremental scan-conversion of a few adds per pixel
  - Triangle visibility is constant (no self-occlusions)
  - Normals can be used to convey flat or curved shading

Triangles

- Poor Modeling primitive:
  - Curved geometries require lots of triangles
  - Continuity/smoothness/blending is difficult
  - Shape design is hard since influence is completely local
  - Inside/outside tests are difficult for non-convex objects

Implicit Modeling Primitives

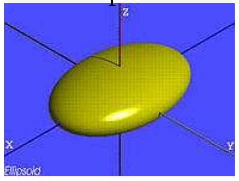
- Expressed by equations of the form
  $f(x, y, z) = 0$
- Divide the space into inside/outside based on whether $f(x, y, z) < 0$ *or* > 0
- Given an object, it is difficult to derive its implicit representation (there has been some recent progress in this direction though)
- The class of known implicit functions is large enough to serve as a useful modeling primitive

Quadrics

- $x^2 + y^2 + z^2 - 1 = 0$ *Sphere/Ellipsoid*
- $x^2 + y^2 - z^2 - 1 = 0$ *Hyperboloid of one sheet*
- $x^2 - y^2 - z^2 - 1 = 0$ *Hyperboloid of two sheets*
- $x^2 + y^2 - z^2 = 0$ *Elliptic Cone*
- $x^2 + y^2 - z = 0$ *Elliptic paraboloid*
- $-x^2 + y^2 - z = 0$ *Hyperbolic paraboloid*

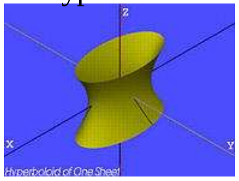
Quadrics

Ellipsoid



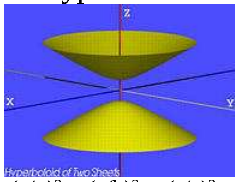
$(x/a)^2 + (y/b)^2 + (z/c)^2 = 1$

Hyperboloid



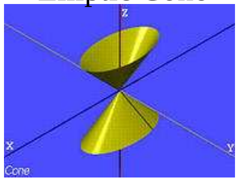
$(x/a)^2 + (y/b)^2 - (z/c)^2 = 1$

Hyperboloid



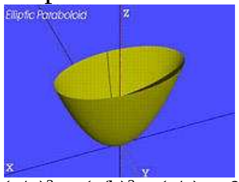
$-(x/a)^2 - (y/b)^2 + (z/c)^2 = 1$

Elliptic Cone



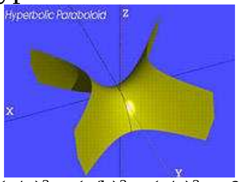
$(x/a)^2 + (y/b)^2 - (z/c)^2 = 0$

Elliptic Paraboloid



$(x/a)^2 + (y/b)^2 - (z/c) = 0$

Hyperbolic Paraboloid



$(x/a)^2 - (y/b)^2 - (z/c)^2 = 0$

Images courtesy, Ching-Kuang Shene, Michigan Technological University

Slide 11 Lecture 20

Superquadrics

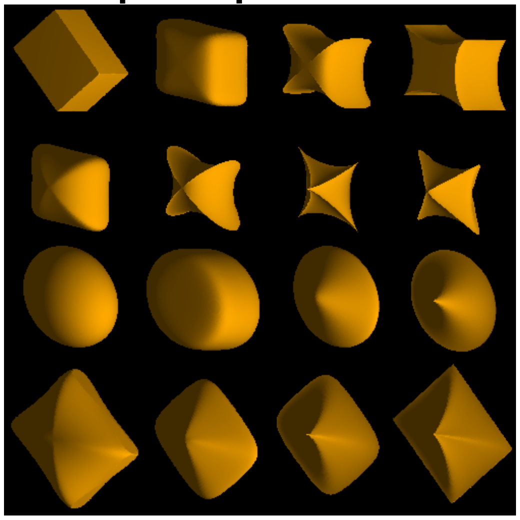


Image Courtesy Montiel, Aguado, Zaluska, University of Surrey, UK

Slide 12 Lecture 20

Superquadrics (Superellipsoids)

- Provide a lot of parameterized flexibility for modeling different kinds of objects

$$\left((x/a)^{2/s} + (y/b)^{2/s} \right)^{s/t} + (z/c)^{2/t} = 1$$

For an ellipsoid, $s = t = 1$

- Similarly one can define superhyperboloids etc…

Slide 13 Lecture 20

Metaballs

- Also known as *blobby models*
- Useful for modeling soft contours: typically muscles for humans, animals
- Have equations of the form:

$$g(x, y, z) = \sum_k b_k f(r - r_k) - T = 0$$

where $f(r)$ is the density function, r_k is the k^{th} center (x_k, y_k, z_k)

- Surfaces are then generated for a given density T

Slide 14 Lecture 20

Metaballs

- Density functions can be exponential:

$$f(r) = e^{-ar^2}$$

or quadratic:

$$f(r) = \begin{cases} b\,(1 - 3r^2/d^2), & 0 < r \le d/3 \\ 3/2\,b\,(1 - r/d)^2, & d/3 < r \le d \\ 0, & r > d \end{cases}$$

Slide 15 Lecture 20

Example

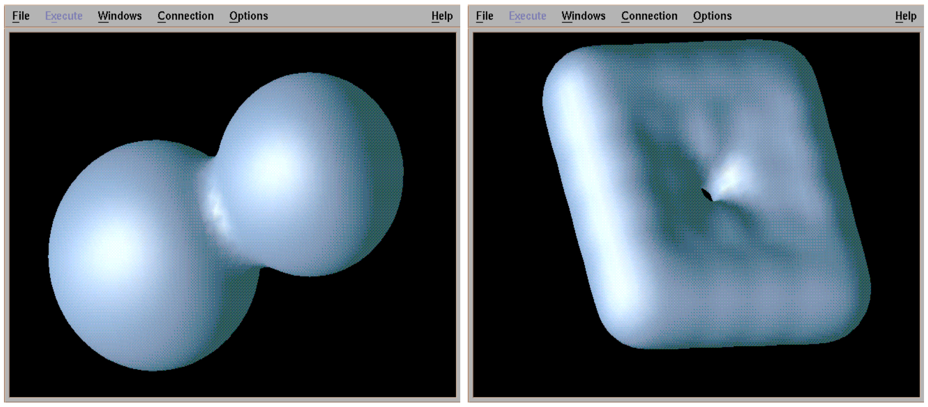
• Suppose we have a single, circularly symmetric Gaussian. What happens as we vary the threshold? We just get a circle, with varying radius.

• Next, suppose we take the sum of two such Gaussians. What happens if we vary the threshold. Do we get 2 circles? Well, at first, when the threshold is high. But as we lower it, the circles join together smoothly. Think of the Gaussians as two hills. As we go from one hill to another, the ground stays higher than when we just leave the hill in another direction.

• Blobby models are good for smoothly joining shapes.

Slide 16 Lecture 20

Metaballs



Images courtesy Matt Ward.

Slide 17 Lecture 20

Metaballs

Image courtesy Spencer Arts

Muscular structure is created using Metareyes plug-in and involved creating hundreds of metaballs.

Slide 18 Lecture 20

Parametric Modeling

- Uses equations of the form:

  $x(u,v) = \dots$

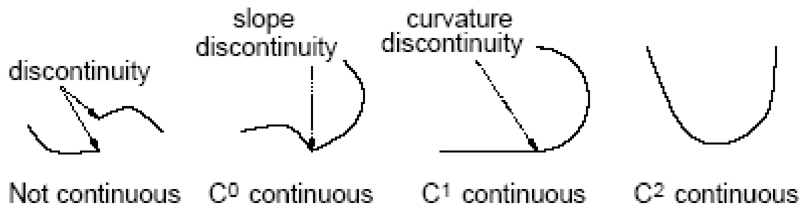
  $y(u,v) = \dots$
- Useful for modeling surfaces where continuity is important
- Allows for trade-off of local versus global influence for editing
- Various representations:

  Bezier, B-splines, NURBS (Non-Uniform Rational B-Splines), trimmed NURBS

Slide 19 Lecture 20

2D Example

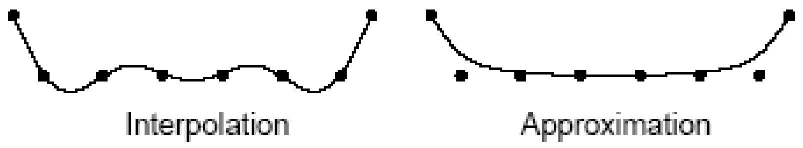
- Use *control points* to specify a curve.
  – Intuitive interface
- Curve should be smooth.



(Dave Mount)

Interpolation vs. Approximation

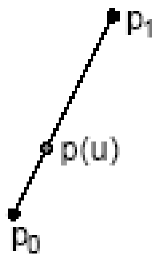
- Smooth curves (eg., polynomials) can be fit to control points.
- But resulting curve can be unpredictable.



(Dave Mount)

Bezier Curves

- Approximate curves
- Two points: $\mathbf{p}(u) = (1-u)\mathbf{p}_0 + u\mathbf{p}_1 \qquad \text{for } 0 \leq u \leq 1.$

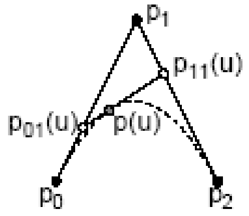


(Dave Mount)

- Three points. Interpolate between pairs.

$$\mathbf{p}_{01}(u) = (1-u)\mathbf{p}_0 + u\mathbf{p}_1$$

$$\mathbf{p}_{11}(u) = (1-u)\mathbf{p}_1 + u\mathbf{p}_2.$$



- Then interpolate between them:

$$\begin{aligned}\mathbf{p}(u) &= (1-u)\mathbf{p}_{01}(u) + u\mathbf{p}_{11}(u)\\ &= (1-u)((1-u)\mathbf{p}_0 + u\mathbf{p}_1) + u((1-u)\mathbf{p}_1 + u\mathbf{p}_2)\\ &= (1-u)^2\mathbf{p}_0 + (2u(1-u))\mathbf{p}_1 + u^2\mathbf{p}_2.\end{aligned}$$

(Dave Mount)

Properties of Bezier Curves

- Can extend to more points
- Each pt on curve is a convex combination of control points.
- Curve starts at first point and ends at last.
- Tangent at 1st (last) point is direction to 2nd (to last).

$$\mathbf{p}(u) = (1-u)^2\mathbf{p}_0 + (2u(1-u))\mathbf{p}_1 + u^2\mathbf{p}_2.$$

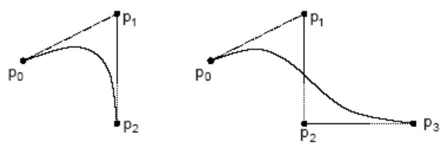
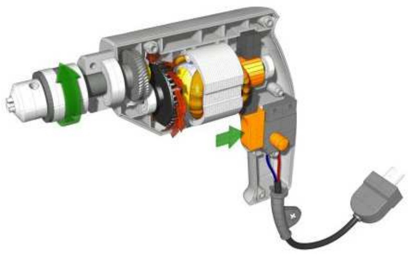


Fig. 77: Bézier curves for three and four control points.

(Dave Mount)

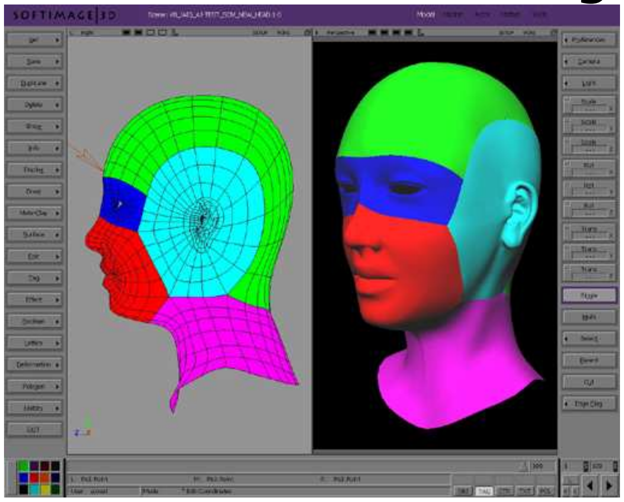
Parametric Modeling



www.rhino3d.com

Slide 25 Lecture 20

Parametric Modeling



Softimage Modeling System

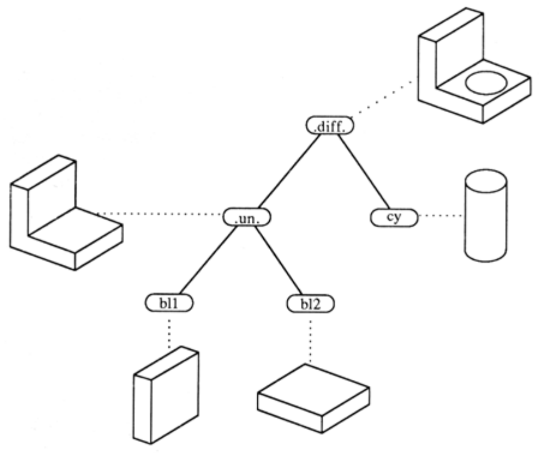
Slide 26 Lecture 20

Constructive Solid Geometry

- Uses primitive objects such as slabs, cylinders, spheres, and cones
- Performs operations of set union, intersection, and difference to build more complex 3D objects from primitives
- Graphics use requires conversion to boundary representation (B-rep) after design
- Very powerful paradigm for design but conversion to B-rep can be challenging

Slide 27 Lecture 20

Constructive Solid Geometry



Slide 28 Lecture 20

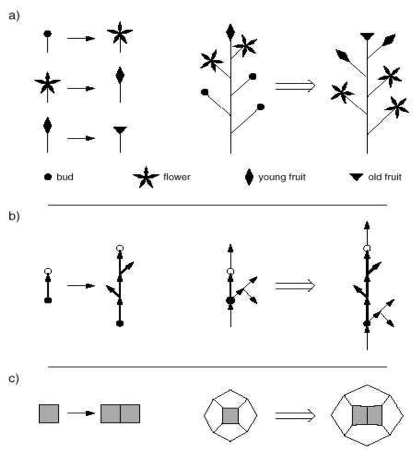
• Determine if a point is inside an object. We do this recursively. Find out whether the point is inside the two parts of the object. Then for union, check if it belongs to either part, for intersection both, for difference one but not other.

• Intersection of object and ray. Recursively determine the range of intersection of the ray and each part. Again, combining is pretty easy.

Procedural Modeling

- Uses procedural rules to *evolve*/*grow* objects
- Iterative application of rules is typical
- Possibilities include:
  - Procedural grammars where rules may be applied deterministically or stochastically
  - Modeling of physical laws of motion and forces, such as attraction/repulsion between particles
  - Iterative evaluation of mathematical functions such as fractals

Procedural Modeling (L-Systems)



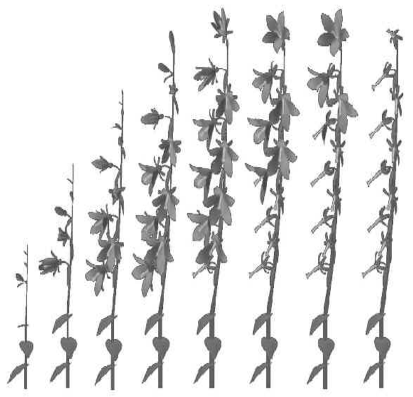


Figure 12: Simulated development of a bluebell flower

Images courtesy, Przemyslaw Prusinkiewicz, University of Calgary, Canada

Slide 31 Lecture 20

Procedural Modeling

Images courtesy, Przemyslaw Prusinkiewicz, University of Calgary, Canada

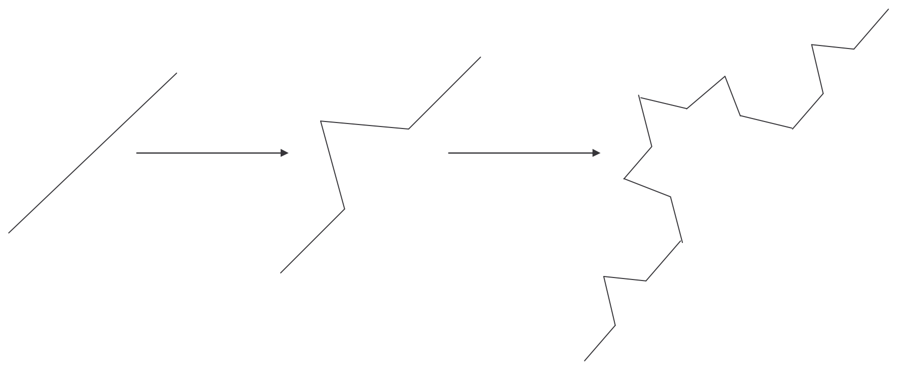
Slide 32 Lecture 20

Fractals

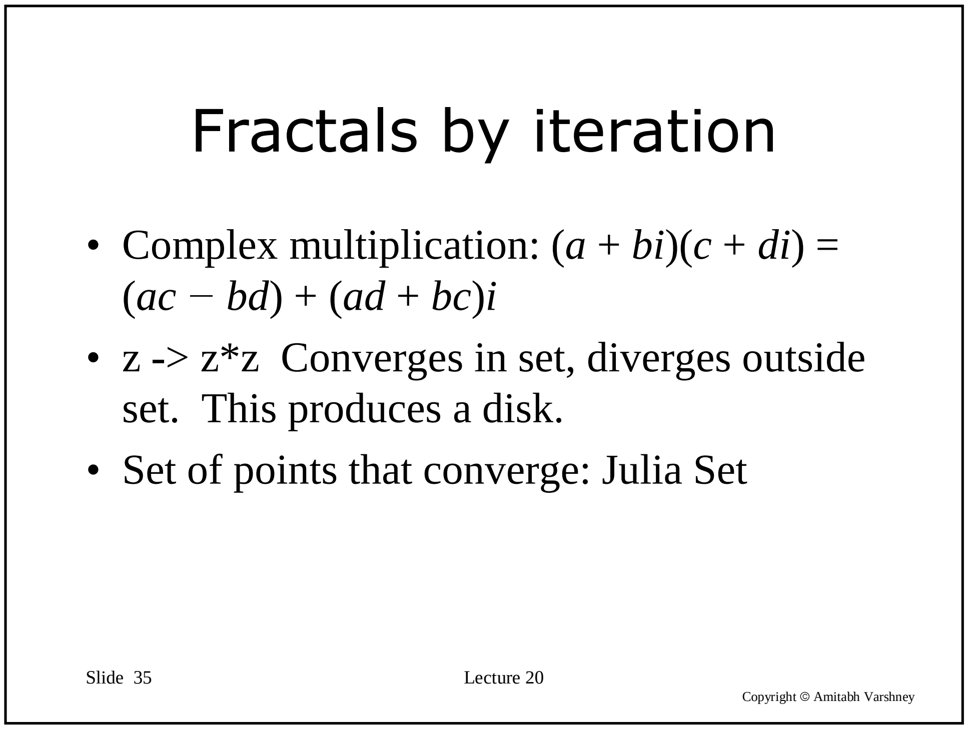
- Shape is self-similar across scales.
- Zooming in, the shape (statistically) looks the same.
- Example: Mountain.
  - – Zoom in and boulders look like mountain.
- Many real objects have fractal appearance.

Slide 33 Lecture 20

Make Fractal



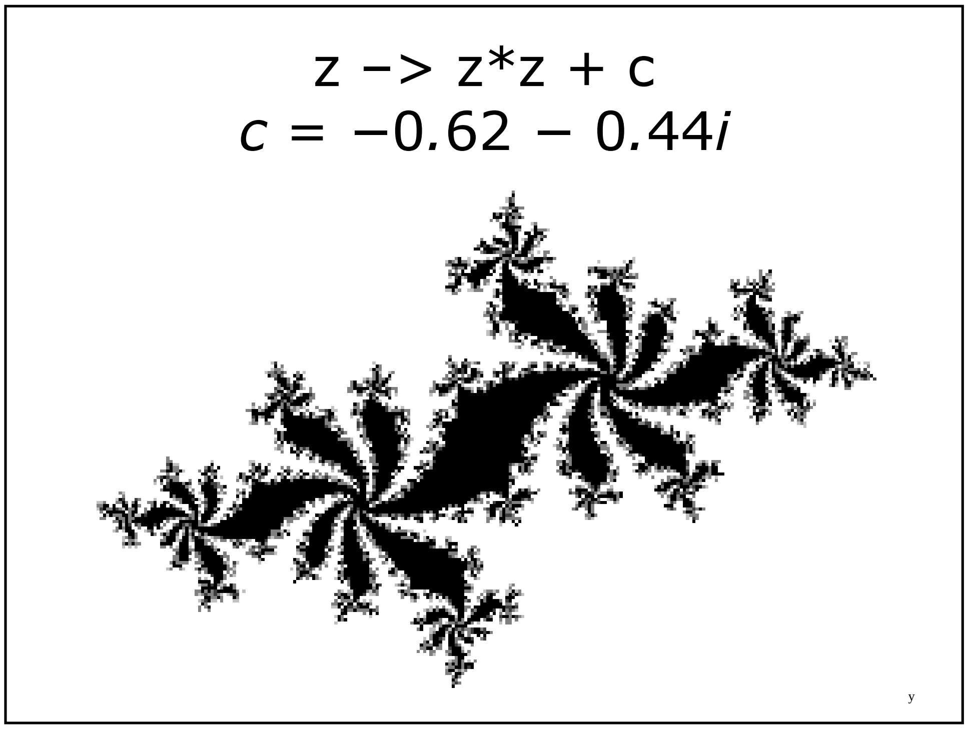
Slide 34 Lecture 20



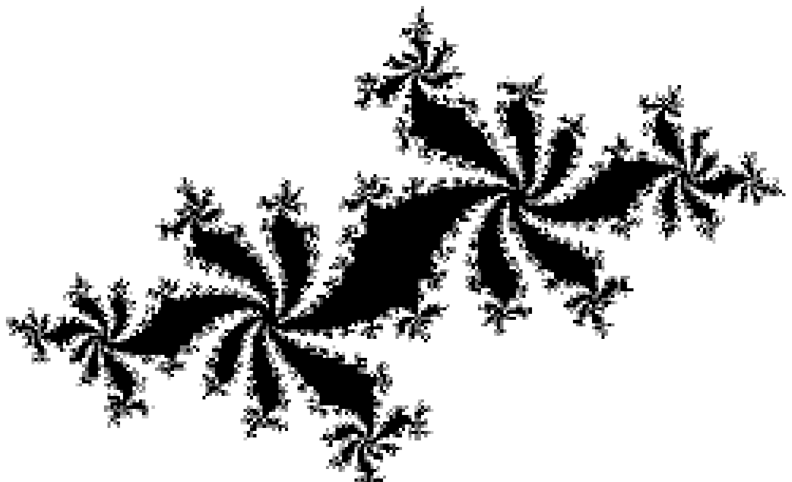
Fractals by iteration

- Complex multiplication: $(a + bi)(c + di) = (ac - bd) + (ad + bc)i$
- z -> z*z Converges in set, diverges outside set. This produces a disk.
- Set of points that converge: Julia Set

Slide 35 Lecture 20



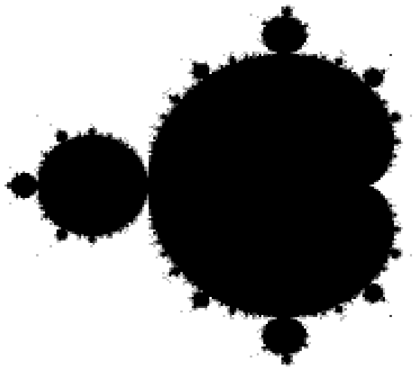
z –> z*z + c
$c = -0.62 - 0.44i$



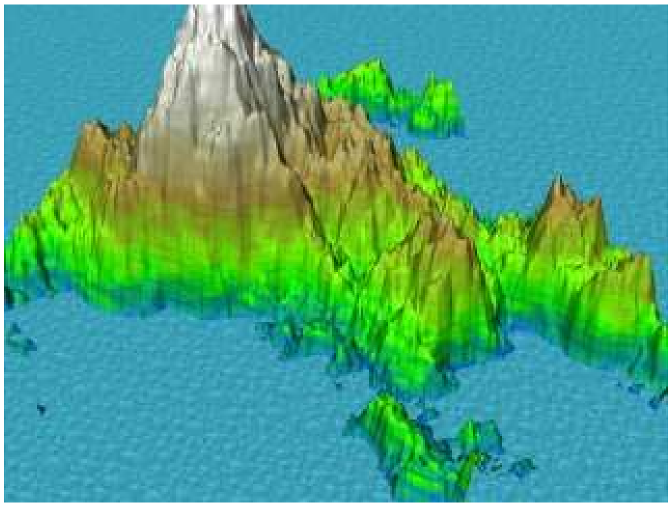
y

Mandelbrot Set

• Set of points, c, for which Julia Set is connected.



Slide 37



•http://www.effectware.com/download/images/efx_mountain2.jpg

Slide 38

Lecture 20

Level of Detail for Polygonal Models

- *Level of detail* or *LOD* methods provide a powerful means for managing scene complexity
- Now a standard tool in graphics to balance rendering speed with visual fidelity

Following slides are from a SIGGRAPH course by Cohen, Huebner, Luebke, Reddy, Varshney, Watson

Motivation

Interactive rendering of large-scale geometric datasets is important

- Scientific and medical visualization
- Architectural and industrial CAD
- Training (military and otherwise)
- Entertainment

Motivation:
Big Models

- The problem:
  - Polygonal models are often too complex to render at interactive rates
- Even worse:
  - Incredibly, models are getting bigger at least as fast as hardware …

Motivation

High complexity: arbitrary topology

Large size: ~ hundreds of millions

Model	Bunny	Happy Buddha	Pieta	St. Matthews Statue	Ste. Pierre Cathedral
Year	1994	1996	1998	2000	2002
# Points	34,947	543,642	7.2 million	127 Million	220 million
Size	200 KB	3 MB	43 MB	762 MB	1.9 GB

Slide 42 Lecture 20

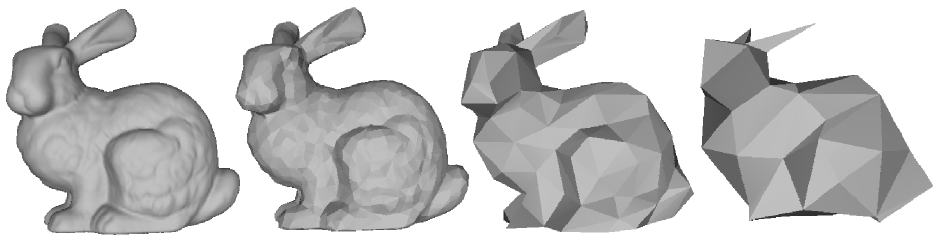
Level of Detail:
The Basic Idea

One solution:

- Simplify the polygonal geometry of small or distant objects
- Known as *Level of Detail* or *LOD*
  - A.k.a. polygonal simplification, geometric simplification, mesh reduction, multiresolution modeling, …

Level of Detail:
Traditional Approach

Create *levels of detail* (LODs) of objects:

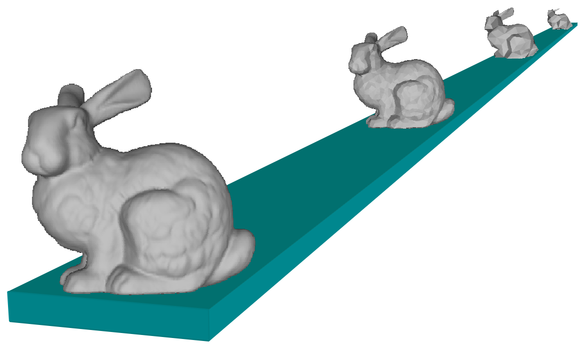


69,451 polys 2,502 polys 251 polys 76 polys

Courtesy Stanford 3D Scanning Repository

Level of Detail:
Traditional Approach

Distant objects use coarser LODs:



Traditional Approach:
Static Level of Detail

- Traditional LOD in a nutshell:
  - Create LODs for each object separately in a preprocess
  - At run-time, pick each object's LOD according to the object's distance (or similar criterion)
- Since LODs are created offline at fixed resolutions, we refer to this as *Static LOD*

Advantages of Static LOD

Simplest programming model; decouples simplification and rendering

- LOD creation need not address real-time rendering constraints
- Run-time rendering need only pick LODs

Advantages of Static LOD

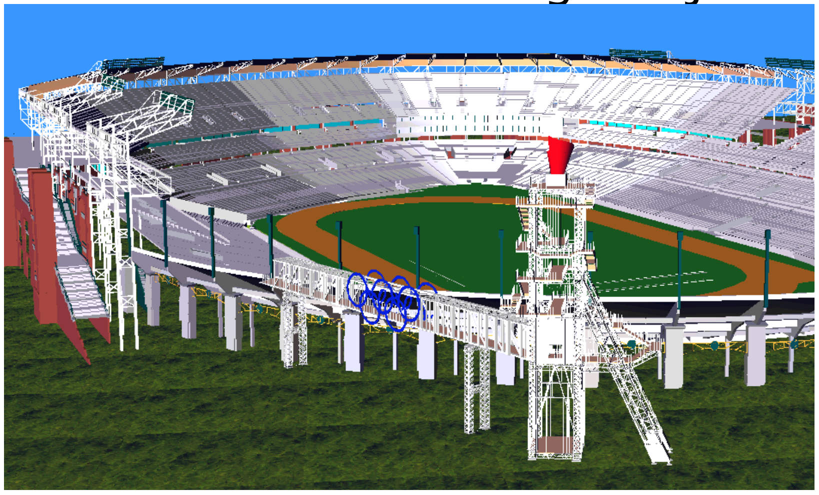
Fits modern graphics hardware well

- Easy to compile each LOD into triangle strips, display lists, vertex arrays, …
- These render *much* faster than unorganized polygons on today’s hardware (3-5 x)

Disadvantages of Static LOD

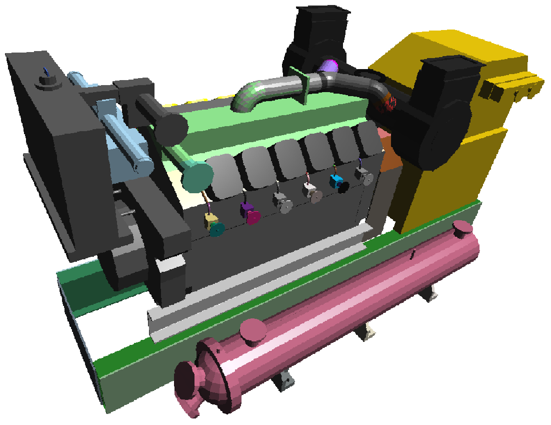
- So why use anything but static LOD?
- Answer: sometimes static LOD not suited for *drastic simplification*
- Some problem cases:
  - Terrain flyovers
  - Volumetric isosurfaces
  - Super-detailed range scans
  - Massive CAD models

Drastic Simplification:
The Problem With Large Objects



Courtesy IBM and ACOG

Drastic Simplification: The Problem With Small Objects



Courtesy Electric Boat

Drastic Simplification

- For drastic simplification:
  - Large objects must be subdivided
  - Small objects must be combined
- Difficult or impossible with static LOD
- *So what can we do?*

Dynamic Level of Detail

A departure from the traditional static approach:

– Static LOD: create individual LODs in a preprocess

– Dynamic LOD: create data structure from which a desired level of detail can be extracted *at run time*.

Dynamic LOD: Advantages

Better granularity means better fidelity

– LOD is specified exactly, not chosen from a few pre-created options

– Thus objects use no more polygons than necessary, which frees up polygons for other objects

– Net result: better resource utilization, leading to better overall fidelity/polygon

Dynamic LOD:
Advantages

Better granularity means smoother transitions

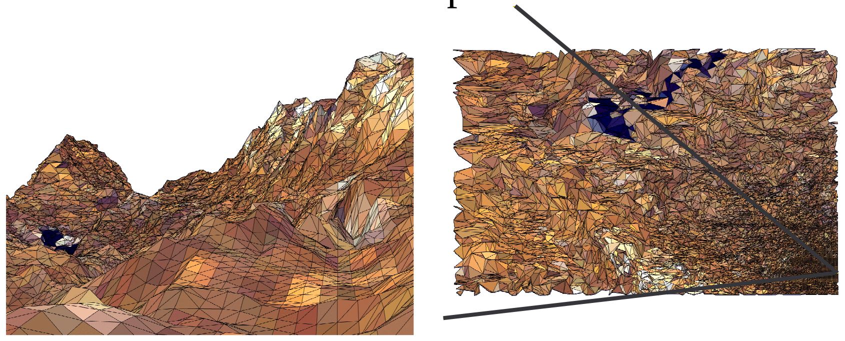
- Switching between traditional LODs can introduce visual "popping" effect
- Dynamic LOD can adjust detail gradually and incrementally, reducing visual pops
  - Can even *geomorph* the fine-grained simplification operations over several frames to eliminate "pops"

Dynamic LOD:
Advantages

- Supports progressive transmission
- Supports *view-dependent LOD*
  - Use current view parameters to select best representation *for the current view*
  - Single objects may thus span several levels of detail

View-Dependent LOD:
Examples

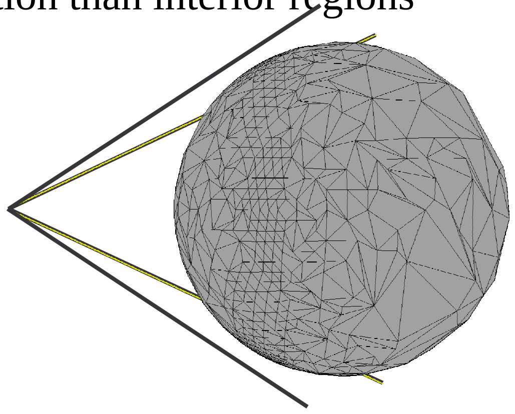
Show nearby portions of object at higher resolution than distant portions



View from eyepoint   Birds-eye view

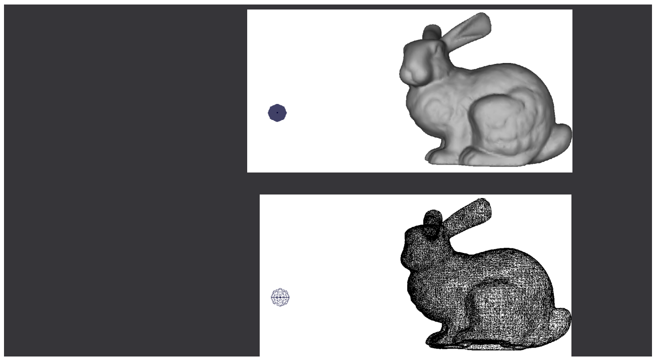
View-Dependent LOD:
Examples

Show silhouette regions of object at higher resolution than interior regions



View-Dependent LOD: Examples

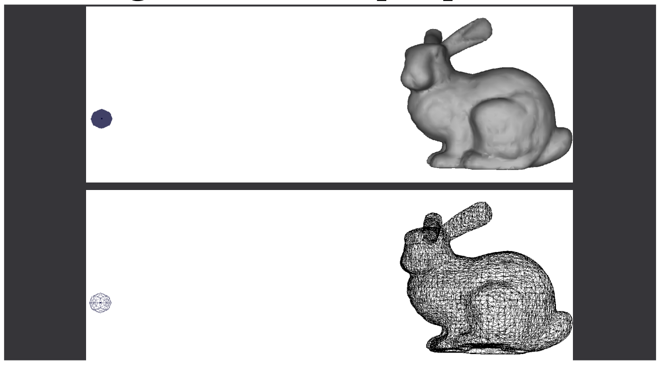
Show more detail where the user is looking than in their peripheral vision:



34,321 triangles

View-Dependent LOD: Examples

Show more detail where the user is looking than in their peripheral vision:



11,726 triangles

View-Dependent LOD: Advantages

- Even better granularity
  - Allocates polygons where they are most needed, within as well as among objects
  - Enables even better overall fidelity
- Enables drastic simplification of very large objects
  - Example: stadium model
  - Example: terrain flyover

View-Dependent LOD: Algorithms

- Many good published algorithms:
  - *Merge Trees* by Xia & Varshney [Visualization 96]
  - *Progressive Meshes* by Hoppe [SIGGRAPH 96, SIGGRAPH 97, …]
  - *Hierarchical Dynamic Simplification* by Luebke & Erikson [SIGGRAPH 97]
  - *Multitriangulation* by DeFloriani et al
  - Others…

Overview

Overview of the algorithm:

- A preprocess builds the *vertex tree*, a hierarchical clustering of vertices
- At run time, clusters appear to grow and shrink as the viewpoint moves
- Clusters that become too small are collapsed, filtering out some triangles