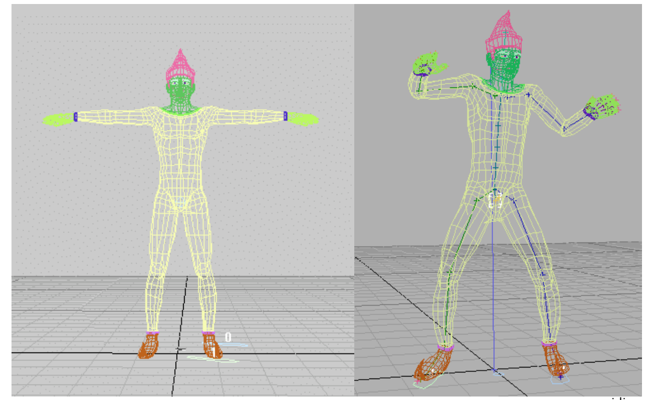
EECS 487: Interactive Computer Graphics

Lecture 40:
- Skinning and rigging, rotation by quaternion
- CSG
- Implicit surfaces, marching cube algorithm

Mesh Skinning

A simple way to deform a surface to follow a skeleton
- simulate skin using a mesh of polygons
- deform mesh based on an underlying skeleton

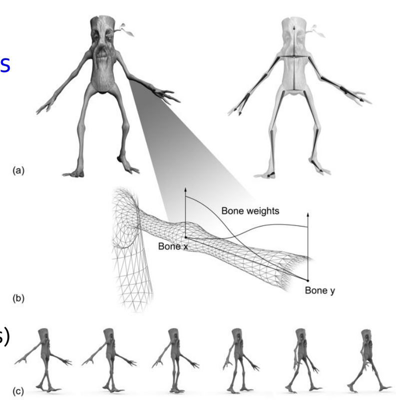


Marschner

nvidia

Linear Blend Skinning

A.k.a. skeletal subspace deformation (SSD), multi-matrix skin, matrix palette skinning, …
- bone rotation deforms space around it
- a vertex on the skin-mesh is attached to multiple nearby bones
- skin deformation is a linear interpolation of transformations on bones: $\mathbf{p}_i' = \sum_j w_{ij} \mathbf{M}_j \mathbf{p}_i$, where
  - $\mathbf{p}_i$: vertex i,
  - $\mathbf{M}_j$ bone j's (rigid) movement transformation matrix
  - w_{ij}: weight/influence of bone j on vertex i ($\neq 0$ only for ≤ 4 nearby bones)



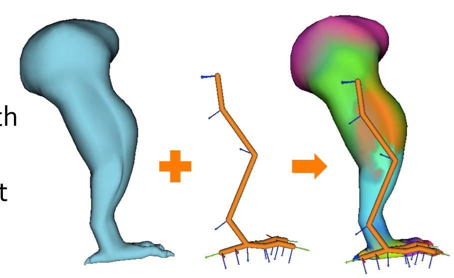
Marschner,Yu

[TP3,Lewis]

Rigging

Weights used in SSD are provided by the animator
- animator can paint weight maps: color-coded influence maps of each bone
- weights can be optimized to match a set of example/bind poses

Rigging:
- associating a bind pose with a skeleton
- and figuring out the weight maps of each bone of the skeleton for the bind pose



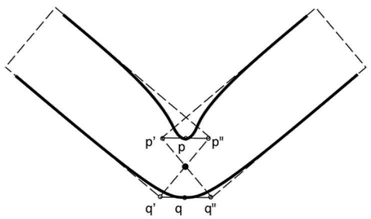
Durand

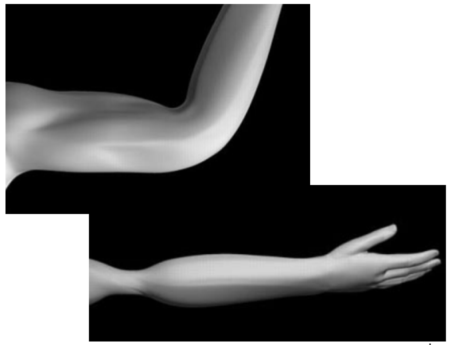
Wang and Phillips 02

SSD Limitation

Surface collapses on the inside of bends and in the presence of strong twists (as when opening a door handle)

- reason: rotations cannot be combined/interpolated linearly!
- solution: add more bones for finer approximation, or change the blending rules (use quaternion)





Marschner

Lewis

Problems with Matrix Representation of Rotation

Doesn't support composition:
90°CW + 90°CCW = zero matrix (instead of identity)

Doesn't allow for interpolation: given rotation matrix $\mathbf{M}_i$ and time t_i, want $M(t)$ such that $M(t_i) = \mathbf{M}_i$

- cannot interpolate each entry independently, for example: let $\mathbf{M}_0$ be identity and $\mathbf{M}_1$ 90° rotation around the x-axis

$$\text{interpolate}\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & -0.5 & 0.5 \end{bmatrix} = \mathbf{M}_{0.5}$$

but $\mathbf{M}_{0.5}$ is not a rotation matrix: it does not preserve rigidity (angles and lengths) and is not orthonormal $\mathbf{M}_{0.5}\mathbf{M}_{0.5}{}^T \neq \mathbf{I}$

Problems with Matrix Representation of Rotation

No composition:
90°CW + 90°CCW = zero matrix (instead of identity)

No interpolation: given rotation matrix $\mathbf{M}_i$ and time t_i, want $M(t)$ such that $M(t_i) = \mathbf{M}_i$

- cannot interpolate each entry independently, for example: let $\mathbf{M}_0$ be identity and $\mathbf{M}_1$ 90° rotation around the x-axis

$$\text{interpolate}\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & -0.5 & 0.5 \end{bmatrix} = \mathbf{M}_{0.5}$$

but $\mathbf{M}_{0.5}$ is not a rotation matrix: it does not preserve rigidity (angles and lengths) and is not orthonormal $\mathbf{M}_{0.5}\mathbf{M}_{0.5}{}^T \neq \mathbf{I}$

Complex Numbers: Review

Vector $\mathbf{v} = [a\ b]^T$ in the complex plane can be written as:

$\mathbf{v} = a + \mathbf{i}\, b$,

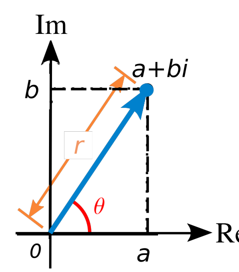
or in polar form:

$\mathbf{v} = r \cos\theta + \mathbf{i}\, r \sin\theta$,

where

$r^2 = a^2 + b^2$ and

$\theta = \tan b/a$



O'Brien, wikipedia

Series Expansion

Recall series expansion of e^x: $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$

Euler: replace x with $\mathbf{i}\theta$

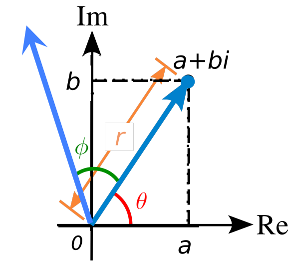
$$
\begin{aligned}
e^{\mathbf{i}\theta} &= 1 + \frac{\mathbf{i}\theta}{1!} + \frac{-\theta^2}{2!} + \frac{-\mathbf{i}\theta^3}{3!} + \frac{\theta^4}{4!} + \ldots \\
&= \left(1 + \frac{-\theta^2}{2!} + \frac{\theta^4}{4!} + \ldots\right) + \mathbf{i}\left(\frac{\theta}{1!} + \frac{-\theta^3}{3!} + \ldots\right) \\
&= \cos\theta + \mathbf{i}\sin\theta
\end{aligned}
$$

Complex Numbers and Rotation

Let $\mathbf{u} = a + \mathbf{i}\,b$ be a complex number
with modulus 1 ($||\mathbf{u}||^2 = a^2 + b^2 = r^2 = 1$),
then $\mathbf{u} = r\cos\theta + \mathbf{i}\,r\sin\theta = re^{\mathbf{i}\theta} = e^{\mathbf{i}\theta}$ for some θ

Pre-multiplying $\mathbf{w} = c + \mathbf{i}\,d = re^{\mathbf{i}\phi}$ with $\mathbf{u}$ gives:
$\mathbf{uw} = (a + \mathbf{i}\,b)(c + \mathbf{i}\,d) = 1e^{\mathbf{i}\theta}re^{\mathbf{i}\phi} = re^{\mathbf{i}(\theta+\phi)}$

$\Rightarrow$ multiplying by a complex number is equivalent to rotation in the 2D plane!

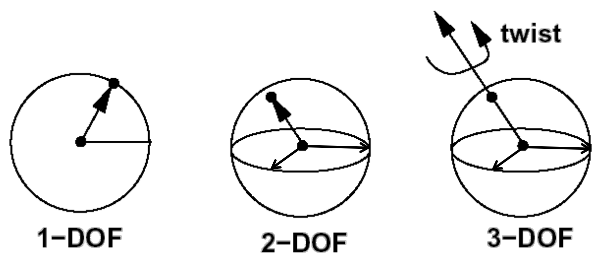


O'Brien, wikipedia

Complex Numbers for 3D Rotation?

By analogy:

- 1-DOF rotations as constrained points on 1D-spheres in 2D
- 2-DOF rotations as constrained points on 2D-spheres in 3D
- 3-DOF rotations as constrained points on 3D-spheres in 4D



Durand

Quaternions

A quaternion is a 4D extension of complex number

Orthonormal basis in quaternions: $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$,
each of which is square root of -1:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$$

Cross-multiplication is like cross product:

$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k}$$
$$\mathbf{ki} = -\mathbf{ik} = \mathbf{j}$$
$$\mathbf{jk} = -\mathbf{kj} = \mathbf{i}$$

Quaternions

A quaternion is a linear combination of $\mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}$:

$$\mathbf{q} = w + x\,\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

$$\|\mathbf{q}\| = \sqrt{w^2 + x^2 + y^2 + z^2}$$

A quaternion can also be interpreted as having a real, scalar part ($s = w$) and an imaginary, vector part $\mathbf{v} = [x\, y\, z]^T$: $\mathbf{q} = (s, \mathbf{v})$

- a real number r is a quaternion with vector $\mathbf{0}$: $\mathbf{q} = (r, \mathbf{0})$
- an ordinary vector $\mathbf{u}$ is a quaternion with scalar 0: $\mathbf{q} = (0, \mathbf{u})$
- a point $\mathbf{p}$ is the quaternion: $\mathbf{q} = (0, \mathbf{p})$
- a quaternion specifies a point in 4D (or 3D if $s = 0$)

Quaternion Properties

Addition and multiplication-by-scalar as usual:

$$\mathbf{q}_1 + \mathbf{q}_2 = (s_1 + s_2,\ \mathbf{v}_1 + \mathbf{v}_2)$$

$$c\mathbf{q} = (cs,\ c\mathbf{v})$$

Multiplication:

- when real parts (s_1, s_2) are zero: $\mathbf{q}_1\mathbf{q}_2 = \left[(-\mathbf{v}_1 \cdot \mathbf{v}_2),\ (\mathbf{v}_1 \times \mathbf{v}_2)\right]$
  the resulting quaternion has a scalar that is a dot product of the two vector parts, negated, and a vector that is their cross product
- if one quaternion has only the scalar part, zero vector part, multiplication is just multiplication by scalar
- combined: $\mathbf{q}_1\mathbf{q}_2 = \left[(s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2),\ (s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)\right]$
- multiplication is associative: $(\mathbf{q}_1\, \mathbf{q}_2)\, \mathbf{q}_3 = \mathbf{q}_1(\mathbf{q}_2\, \mathbf{q}_3)$
  but not commutative: $\mathbf{q}_1\, \mathbf{q}_2 \neq \mathbf{q}_2\, \mathbf{q}_1$

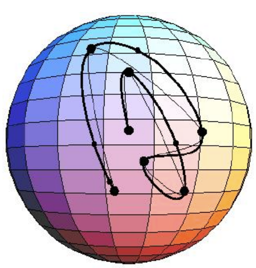
Quaternion Properties

Conjugate: $\mathbf{q}^* = (s, -\mathbf{v}) = w - x\mathbf{i} - y\mathbf{j} - z\mathbf{k}$
$(\mathbf{q}^*)^* = \mathbf{q}$; $(\mathbf{q}_1\mathbf{q}_2)^* = \mathbf{q}_2{}^*\mathbf{q}_1{}^*$; $(\mathbf{q}_1 + \mathbf{q}_2)^* = \mathbf{q}_1{}^* + \mathbf{q}_2{}^*$

Magnitude: $|\mathbf{q}| = \sqrt{\mathbf{q}\mathbf{q}^*} = \sqrt{s^2 + \mathbf{v}\cdot\mathbf{v}}$

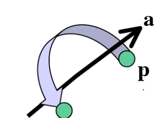
Unit quaternion: $|\mathbf{q}| = 1$

Inverse: $\mathbf{q}^{-1} = \dfrac{\mathbf{q}^*}{|\mathbf{q}|^2}$; $\mathbf{q}^{-1}\mathbf{q} = \mathbf{1}$



for unit quaternion, $\mathbf{q}^{-1} = \mathbf{q}^*$
Unit quaternions form a 3D sphere in the 4D space of quaternions
The product of two unit quaternions is another unit quaternion

Rotation by Unit Quaternion



If $|\mathbf{q}| = 1$ and
$\mathbf{a}$ is a normalized vector through the origin,
$\mathbf{q} = (\cos\theta/2, \sin\theta/2\ \mathbf{a})$ represents a rotation by angle θ about $\mathbf{a}$:
$(0, \mathbf{p}') = \mathbf{q}(0, \mathbf{p})\mathbf{q}^{-1}$, $\mathbf{p}'$ is $\mathbf{p}$ rotated by θ about $\mathbf{a}$

$\mathbf{q}$ can also be written as $\mathbf{q} = \cos\theta/2 + \mathbf{a}\sin\theta/2 = re^{\mathbf{a}\theta/2}$

Any quaternion $\mathbf{q} = re^{\mathbf{a}\theta/2}$ can be interpreted as a rotation simply by normalizing it (dividing by its length)
⇒ for $\mathbf{q} = (s, \mathbf{v})$, there exists a vector $\mathbf{a}$ and a θ such that:
$\mathbf{q} = (\cos\theta/2, \sin\theta/2\ \mathbf{a})$

Both $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation (corresponding to angles θ and $(2\pi - \theta)$)

Rotation by Unit Quaternion

$\mathbf{q} = (s, \mathbf{v}) = (\cos\,\theta/2, \sin\,\theta/2\;\mathbf{a}),\ ||\mathbf{a}|| = 1$

Rotation of $\mathbf{u}$ around $\mathbf{a}$ can be computed as: $\mathbf{q}(0, \mathbf{u})\mathbf{q}^{-1}$

Using the quaternion multiplication rule:

$$\mathbf{q}_1\mathbf{q}_2 = \left[(s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2), (s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)\right]$$

we find that the scalar part of the result is 0, and the vector part is:

$$= (s^2 - \mathbf{v}\cdot\mathbf{v})\mathbf{u} + 2\mathbf{v}(\mathbf{v}\cdot\mathbf{u}) + 2s(\mathbf{v}\times\mathbf{u})$$
$$= (\cos^2(\theta/2) - \sin^2(\theta/2))\mathbf{u}$$
$$+ 2\sin^2(\theta/2)\mathbf{a}(\mathbf{a}\cdot\mathbf{u})$$
$$+ (2\cos(\theta/2)\sin(\theta/2))(\mathbf{a}\times\mathbf{u})$$
$$= \boxed{\cos\theta\mathbf{u} + (1-\cos\theta)\mathbf{a}(\mathbf{a}\cdot\mathbf{u}) + (\sin\theta)(\mathbf{a}\times\mathbf{u})}$$ which is . . .

Using:
$\mathbf{r}\times\mathbf{s} = -\mathbf{s}\times\mathbf{r}$
$\mathbf{r}\times\mathbf{s}\times\mathbf{t} = (\mathbf{r}\cdot\mathbf{t})\mathbf{s} - (\mathbf{r}\cdot\mathbf{s})\mathbf{t}$
$\cos(2\varphi) = \cos^2\varphi - \sin^2\varphi$
$\sin^2(\varphi/2) = (1-\cos\varphi)/2$
$\sin(2\varphi) = 2\sin\varphi\cos\varphi$

Rotation by Unit Quaternion

Advantages of quaternions:

- more compact than rotation matrices
- can be easily converted to matrices if necessary, for $\mathbf{q} = (\cos\,\theta/2, \sin\,\theta/2\;\mathbf{a}) = (w, \mathbf{v}) = (w, x, y, z)$

$$\mathbf{R}_\mathbf{q} = \begin{pmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy & 0 \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx & 0 \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Can compose rotations by quaternion multiplication, e.g., two rotations $\mathbf{q}_1$ and $\mathbf{q}_2$ composed as:

$\mathbf{q}_2\mathbf{q}_1(0, \mathbf{u})\mathbf{q}_1^{-1}\mathbf{q}_2^{-1}$

Linear Interpolation

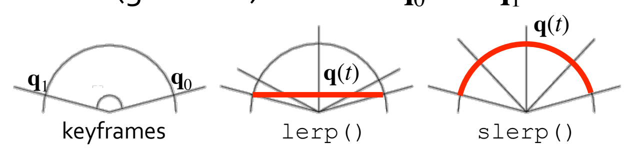
Using linear interpolation (`lerp()`) to interpolate between 2 orientations (i.e., quaternions):

$$\mathtt{lerp}(\mathbf{q}_0, \mathbf{q}_1, t) = \mathbf{q}(t) = (1-t)\mathbf{q}_0 + t\mathbf{q}_1$$

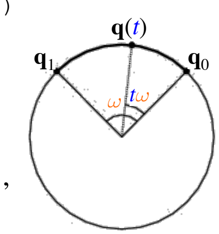
results in a straight line progressions of interpolated orientations with non-uniform velocity (accelerates towards the middle, equidistance time intervals correspond to non-equidistant arc lengths):

Spherical Linear Interpolation

Spherical linear interpolation (`slerp()`) interpolates on the surface of the 4D unit hypersphere along the great arc (geodesic) between $\mathbf{q}_0$ and $\mathbf{q}_1$



The usual trigonometric rules hold on the 4D arc, and `slerp()` is given by:



$$\mathbf{q}(t) = \mathtt{slerp}(\mathbf{q}_0, \mathbf{q}_1, t) = \frac{\sin(1-t)\omega}{\sin\omega}\mathbf{q}_0 + \frac{\sin t\omega}{\sin\omega}\mathbf{q}_1,$$

$\omega = \cos^{-1}(\mathbf{q}_0 \cdot \mathbf{q}_1) = \frac{\theta}{2}$, the angle between $\mathbf{q}_0$ and $\mathbf{q}_1$

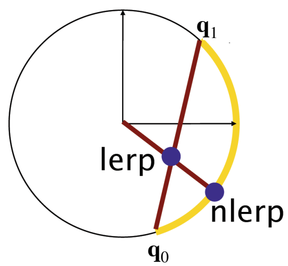
nlerp()

Linearly-interpolated quaternions are not unit quaternions

But slerp() is expensive: lots of sine evaluations

nlerp(): do linear interpolation and then normalize

- not uniform, but not so bad if rotations are close enough



For details see:
http://number-none.com/product/Hacking%20Quaternions/index.html

Durand

Smooth Spherical Curves

With slerp()

- rotation interpolates smoothly between two orientations
- is straightforward to compute
- no gimbal lock
- "twisting" motion is not an issue

but

- consecutive rotations around different axes (all passing through a common point) produce sharply changing motion

Smooth spherical curves are similar to splines but uses spherical linear interpolation instead of simple linear interpolation [Shoe85]

TP3

Constructive Solid Modeling (CSG)

Widely (mainly) used in CAD/CAM

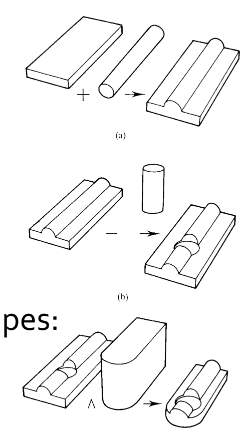
Object modeling for:

- casting, machining, extruding, etc.

Many manufactured objects can be represented by "combinations" of elementary geometric primitives

Primitives consist of rigid geometric shapes:

- blocks, pyramids, cylinders, spheres, etc.

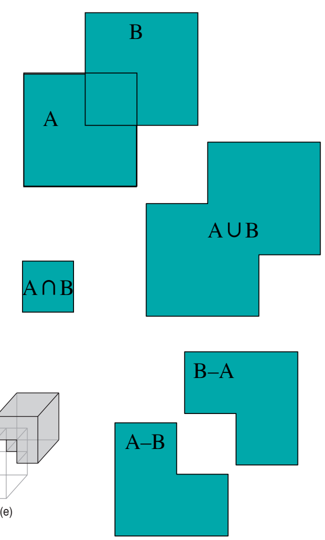


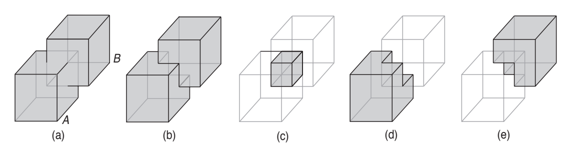
Watt 2000

Constructive Solid Geometry

Construct complex shapes by combining simple primitives using boolean set operations

- union: $A \cup B$, $A + B$, A or B
- intersection: $A \cap B$, $A * B$, A and B
- subtraction: $A \backslash B$, $A - B$, A and not B

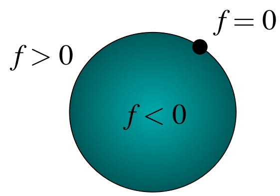




Foley, van Dam 92

Hart08

Implicit Surfaces



Use the implicit functions of surfaces to represent objects

- actually describe solids since they have well-defined inside and outside

Examples:

- sphere
- ellipsoid
- torus
- paraboloid
- hyperboloid

Implicit function is polynomial:

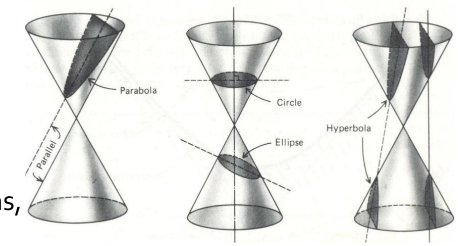
$f(x,y,z) = ax^d + by^d + cz^d + ex^{d-1}y + fx^{d-1}z + gy^{d-1}z + \ldots$

Funkhouser

Conic Sections

A common class of curves with a very long history

- defined by intersections of a plane with a cone
- describes several generally useful kinds of curves: circles, ellipses, parabolas, hyperbolas, and lines
- defined implicitly by the quadratic polynomial $f(x,y) = ax^2 + 2bxy + 2cx + dy^2 + 2ey + f = 0$
- in matrix form:



$$f(x,y) = \mathbf{p}^T\mathbf{Q}\mathbf{p} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

Yu, andrews.edu

Quadric Surfaces

Quadrics are 3D analogues of conics

- defined by quadratic polynomial:

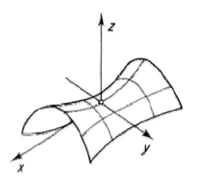
$f(x,y,z) = ax^2 + 2bxy + 2cxz + 2dx + ey^2 + 2fyz + 2gy + hz^2 + 2iz + j = 0$

- or in matrix form:

$$f(x,y,z) = \mathbf{p}^T\mathbf{Q}\mathbf{p} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ d & g & i & j \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

- unit surface normal:

$$\mathbf{n} = \frac{\nabla f}{\|f\|}, \text{ where } \nabla f = \begin{bmatrix} \partial f/\partial x \\ \partial f/\partial y \\ \partial f/\partial z \end{bmatrix}$$

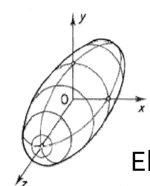


Hyperbolic paraboloid

Yu,Heckbert

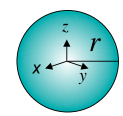
Quadrics

Variations?
Use the transformation trick



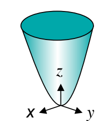
Ellipsoid:

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 - 1 = 0$$



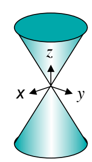
Sphere:

$x^2 + y^2 + z^2 - r^2 = 0$



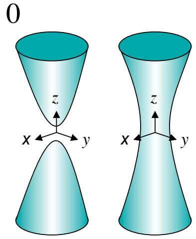
Paraboloid:

$x^2 + y^2 - z = 0$



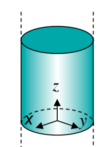
Cone:

$x^2 + y^2 - z^2 = 0$



Hyperboloid:

$x^2 + y^2 - z^2 \pm r^2 = 0$

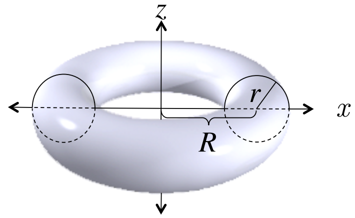


Cylinder:

$x^2 + y^2 - r^2 = 0$

Heckbert,Hart,Funkhouser

Torus



Product of two implicit circles:

$(x - R)^2 + z^2 - r^2 = 0$ and

$(x + R)^2 + z^2 - r^2 = 0$

$$((x - R)^2 + z^2 - r^2)((x + R)^2 + z^2 - r^2)$$
$$= (x^2 - 2Rx + R^2 + z^2 - r^2)\,(x^2 + 2Rx + R^2 + z^2 - r^2)$$
$$= x^4 + 2x^2z^2 + z^4 - 2x2r^2 - 2z2r^2 + r^4 - 2x^2R^2 + 2z^2R^2 - 2r^2R^2 + R^4$$
$$= (x^2 + z^2 - r^2 - R^2)^2 + 4z^2R^2 - 4r^2R^2$$

Surface of rotation about the z-axis: replace x^2 with x^2+y^2

$$f(x,y,z) = (x^2 + y^2 + z^2 - r^2 - R^2)^2 + 4R^2(z^2 - r^2)$$

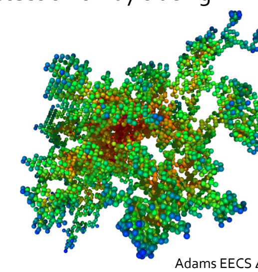
Harto8

Advantages of Implicit Surface

The entire surface is represented by a single, constant value function (a.k.a. a level set or isosurface of the function)
- compact
- cleanly defined solid along with its boundary: can guarantee that model is watertight

Easy to determine if a point lies on the curve
- inside/outside test: great for intersections, unions, subtractions (CSG)
- useful as bounding volumes, e.g., for collision detection or ray tracing

Easy to compute surface normal



Efficient topology changes: can handle weird topology for animation

Model some real medical/scientific data well

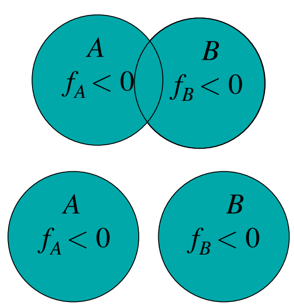
Hodgins,Ramamoorthi,Yu,Durand

Adams EECS 487 F09

CSG on Implicit Surfaces

Boolen operations are replaced by arithmetic:
- MAX replaces AND (intersection)
- MIN replaces OR (union)
- MINUS replaces NOT (unary subtraction)

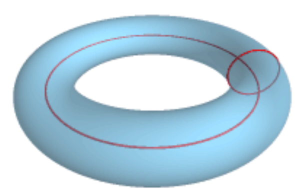
Thus:
- $f_{A\cup B} = \mathrm{MIN}(f_A, f_B)$
- $f_{A\cap B} = \mathrm{MAX}(f_A, f_B)$
- $f_{A-B} = \mathrm{MAX}(f_A, -f_B)$

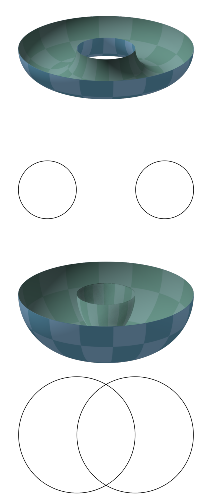


Hodgins

Efficient Topology Changes

As the distance to the axis of revolution decreases, the ring torus becomes a spindle torus and then degenerates into a sphere





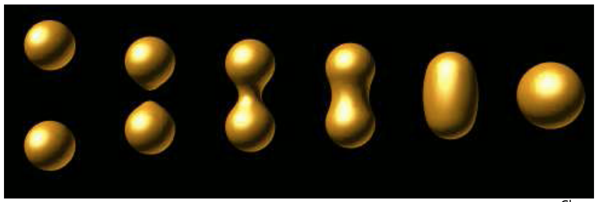
wikipedia

"Blobby" Models

Another advantage of isosurfaces is that you can add them up to merge the shapes!

Blobby model defines implicit surface as combination of blobs or metaballs

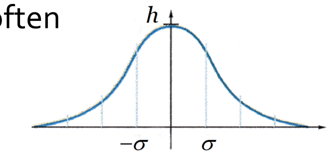
- each blob is formed from a seed point, $\mathbf{s}_i$
- each seed point has a potential field surrounding it
- when the potential fields of two blobs overlap, they merge to form an implicit surface soft object



Chenney.Yu,Hodgins,Durand

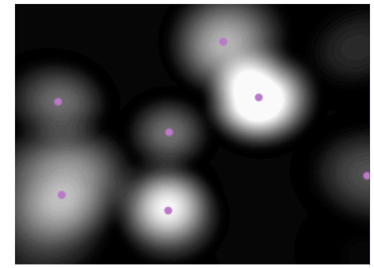
"Blobby" Models

The potential field of a blob is usually an exponential function of distance (of surface point $\mathbf{p}$) from the seed, $f(\mathbf{p}, \mathbf{s}_i)$; often Gaussian is used:



$$f_i(\mathbf{p},\mathbf{s}_i) = h_i e^{-\sigma_i\|\mathbf{p}-\mathbf{s}_i\|^2} - \tau$$

- varying the standard deviations (σ_i's) of the Gaussians makes each blob bigger
- varying the threshold (τ) makes blobs merge or separate



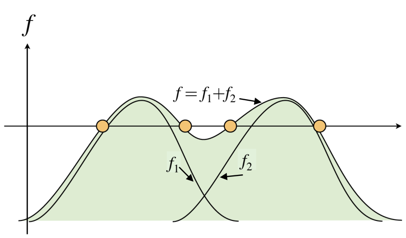
Chenney.Yu,Hodgins,Durand

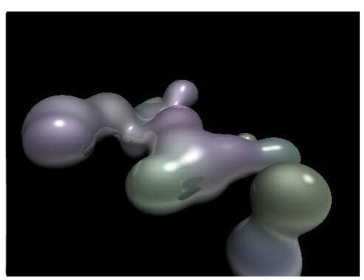
"Blobby" Models

The implicit surface soft objects is a combination of these functions :

$$f(\mathbf{p}) = \left(\sum_i w_i f_i(\mathbf{p},\mathbf{s}_i)\right) - \tau = 0$$

Example isosurface of a 3D function:





Yu,Durand

Disadvantages of Implicit Surface

Fitting to real world data is not easy

- no sharp edges
- function extends to infinity, must trim to get desired patch (not easy)

Interactive control is not easy

Terrible for iterating over surface (unlike parametric)

⇒ expensive to render

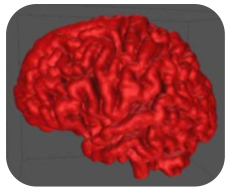
- ray tracing is easiest (easier than parametric)
- can also use parametric surfaces (NURBs)
- or convert to polygons: Marching Cubes algorithm

Hodgins,Ramamoorthi,Funkhouse

Marching Cubes Algorithm

Used to convert an implicit surface to a polygonal mesh, for rendering by hardware, for example

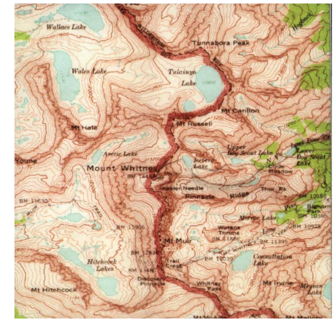
Also used to render isosurface of volumetric data:
- function defined by regular samples on a 3D grid (like an image, but in 3D)
- example uses: medical imaging, numerical simulation, scientific visualization in general



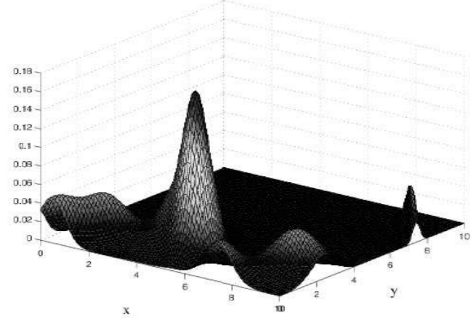
James

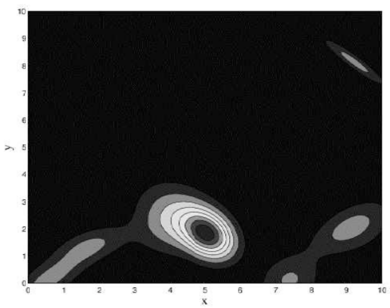
Topics:
- height maps and contour curves
- drawing 2D isocontours using marching squares
- drawing 3D isosurfaces using marching cubes

Height Maps and Contour Plots



Height maps represent various data values with levels of elevation
- can be represented as contour-curves in 2D

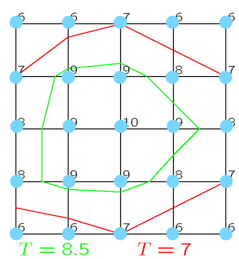




Spadaccini07

Contour Tracking and Drawing

Contour marks the boundaries between regions of different scalar values
- can be lines (isocontours) in 2D, or
- surfaces (isosurfaces) in 3D

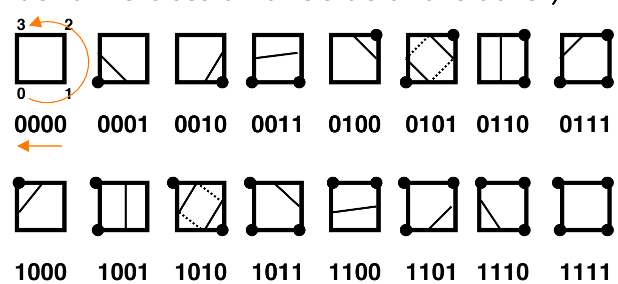


Consider the sample grid
- at every grid point (x_i, y_i), $f_{ij} = f(x_i, y_i)$ is either $\leq$ or $\geq c$
- the red contour shows where $f(x,y)$=7, the green one, $f(x,y)$=8.5
- these contours can be constructed by linear interpolation:
  - e.g., 7 is ⅓-way between (6, 9) and ½-way between (6, 8)
  - connect the intersection points
  - simple, fast, usually sufficient

Spadaccini07

Marching Squares

Observation: there is only a finite number of ways a contour can pass through a cell (topological states,* number of vertices on one side or the other) :

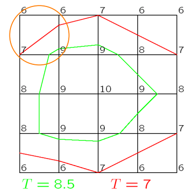


- $f(x,y)$ represented by the line
- black dots represent vertices with value $> f(x,y)$
- (we're not computing intersections, yet)

*we don't need to store all 16 states, instead transform similar states

Lorensen&Cline87

Marching Squares



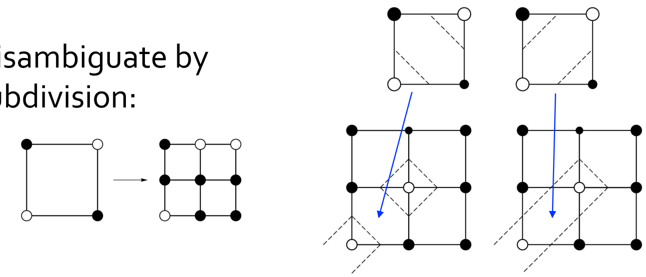
To draw a contour:

1. For each cell, compute the inside/outside state of each vertex given the implicit function, e.g., $f(x,y)=7$
2. Generate an index from the lower left vertex, ccw, e.g., 0010
3. Look up topological state to determine contour "type"
4. Place contour on edges by interpolating between the values of its two vertices, e.g., 7 between $(6, 9)$
5. Contour plot can be generated by "marching" through the grid, left to right, top to bottom (with additional details . . .)

Marching Squares Ambiguities

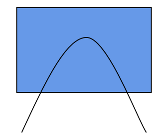
Ambiguous labels can result in different contours

Disambiguate by subdivision:



Straddling cells: at least one vertex inside and one outside surface

- non-straddling cells can still contain contour

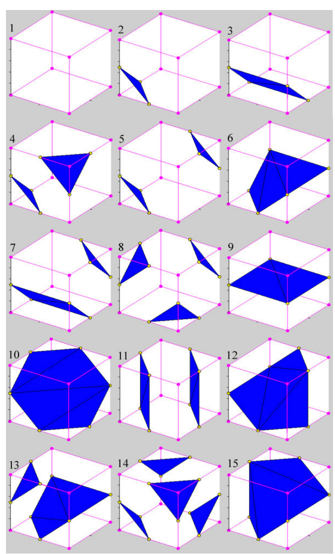


Marching Cubes

Marching square extended to $3D$

Used to create isosurfaces (contours in $3D$)

Contour passes through a cell in one of 15 topological states:



Marching Cubes

To render an implicit surface:

1. put object inside a $3D$ grid of cells
2. each vertex of a cell is either $>$ or $<$ the value of the isosurface at that cell
3. classify each cell into one of the 15 topological states
4. interpolate edge intersection from vertex values
5. build connectivity
6. be careful with correct orientation of surface normal and state/label ambiguity