Motion

CS4451

Prof. Jarek Rossignac

College of Computing

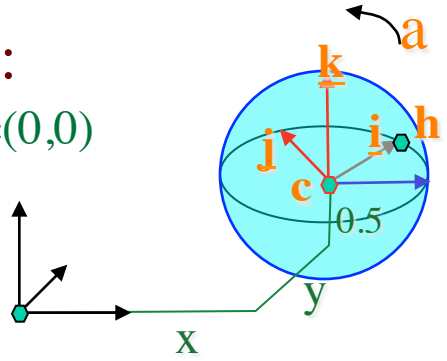
Georgia Institute of Technology

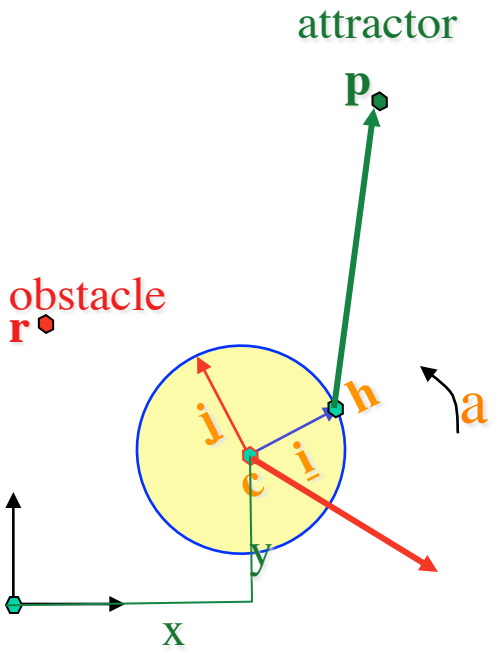
Walking bugs

- Bug body moves and rotates (2D motion at constant height)
- Feet perform cyclic walking motion (uncoordinated)
- Bug's head is attracted by spring force to nearby attractors
  - Attraction increases with distance, until a limit (spring brakes)
- Bugs body is repelled by nearby bugs and trees
  - Repulsion decreases with distance and stays 0 when far
- Behavior
  - Aggressive bug tries to attack the head of nearby bugs
  - Submissive bug tries to follow the tail of nearest bug in vicinity
  - Sumo bug: tries to push other bugs out of the ring
    - Keep attraction constant towards the center of closest-by bug
    - Give it more mass than the bug you control
    - Try to push it out of a circular ring
    - Try fighting several sumo bugs at once

Moving a bug with naïve physics

- Bug is initially in a ball of radius 1 with center c=(0,0,0.5)
  - Its head h is initially at (1, 0, 0.5)
- Bug current position is defined by parameters: x, y, and a
  - Translate by vector (x,y,0.5); Rotate around vertical by angle a
- Change its 2D translational velocity (x', y'):
  - (x', y') += $R_5(rc) + A_3(hp)$;
    - $R_d(\underline{\mathbf{V}}) := (d - \|\underline{\mathbf{V}}\|)\underline{\mathbf{V}}_u$ when $\|\underline{\mathbf{V}}\| < d$ and $\underline{\mathbf{0}}$ otherwise
    - $A_d(\underline{\mathbf{V}}) := \|\underline{\mathbf{V}}\|\underline{\mathbf{V}}_u$ when $\|\underline{\mathbf{V}}\| < d$ and $\underline{\mathbf{0}}$ otherwise
- Change its rotational velocity a':
  - a' += $A_3(hp) \bullet \underline{j}$
- You may also need friction:
  - IF $\|(x', y')\| < v$ THEN (x', y') =(0,0)
    ELSE (x', y') –=$v(x', y')_u$;
  - IF $|a'| < a_0$ THEN a' = 0
    ELSE a' –= a_0 a'/|a'|
- Assume c, h, p, r have same height

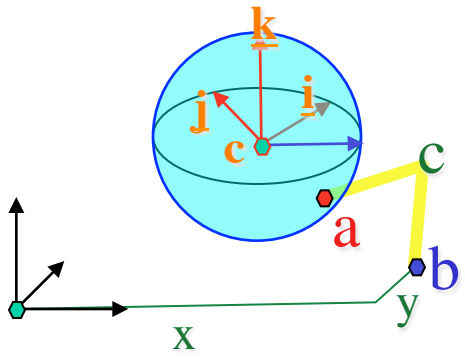


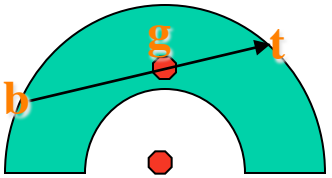


Animating the feet

Walk sequence has two states:

- Toe-down state:
  - keep b fixed in on the floor as the bug moves
  - Compute b in local coordinate system of hip
  - Check b against reachable zone
  - If b is out of the reachable zone, Z, then set target t as far as possible in Z behind the center, g, of Z, set up the parameters for toe-up motion and switch to toe up state
- Toe-up state:
  - Update 3D position of b in the local coordinates of the hip
    - **Initially try using a linear constant velocity horizontal motion combined with a vertical ballistic motion (constant gravity)**
    - **Then explore mappings that smoothen out the motion**
  - When b has reached t, then switch back to toe-down state

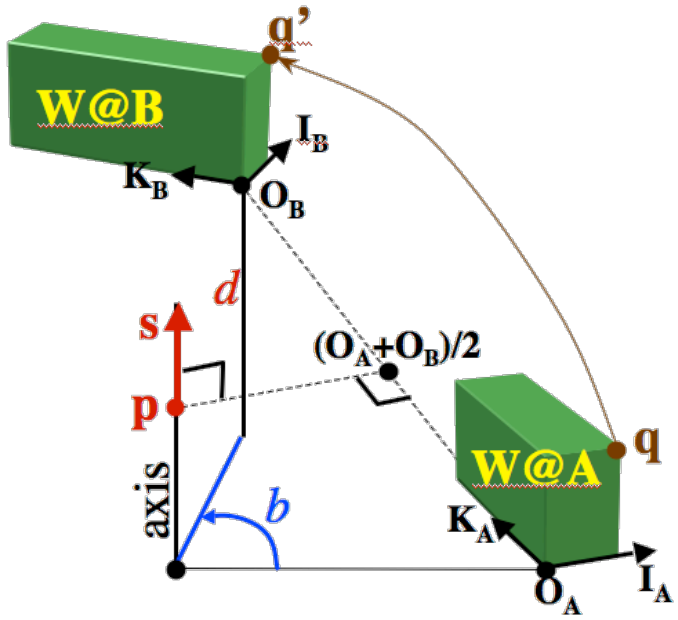


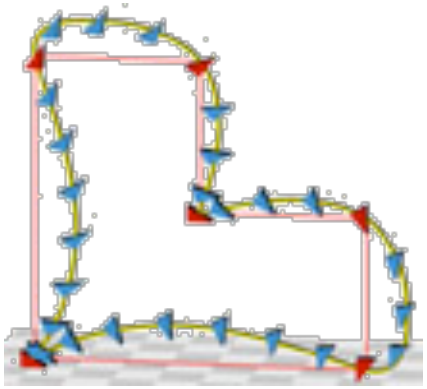


Motion smoothing

Georgia Tech College of Computing
Interactive & Intelligent Computing

- Polyscrew interpolating motions
- Jarek subdivisions applied to polyscrews



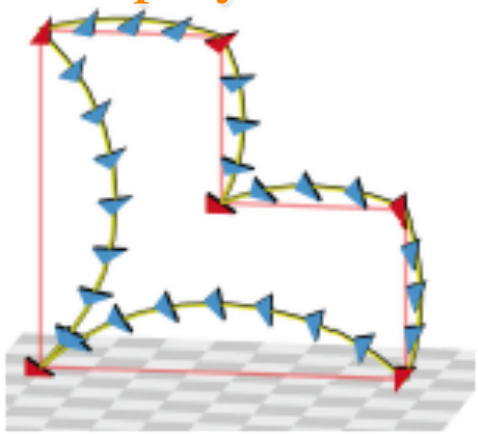


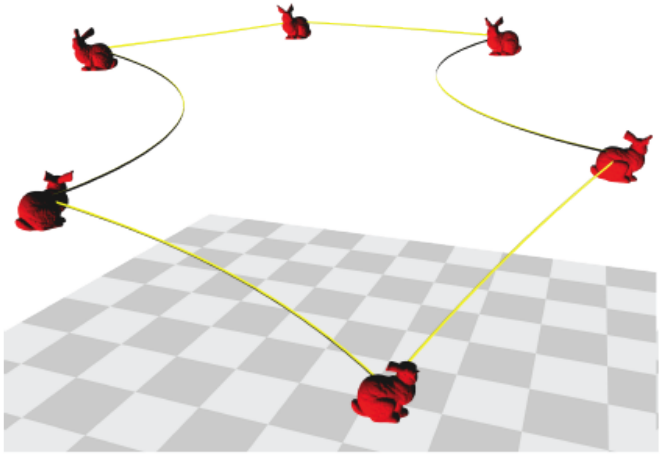
Updated November 9, 2012

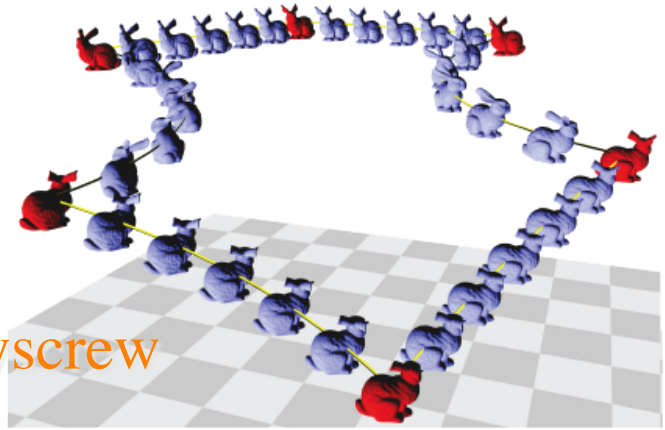
How to interpolate key poses?

Polyscrew: A screw between each pair of consecutive key poses

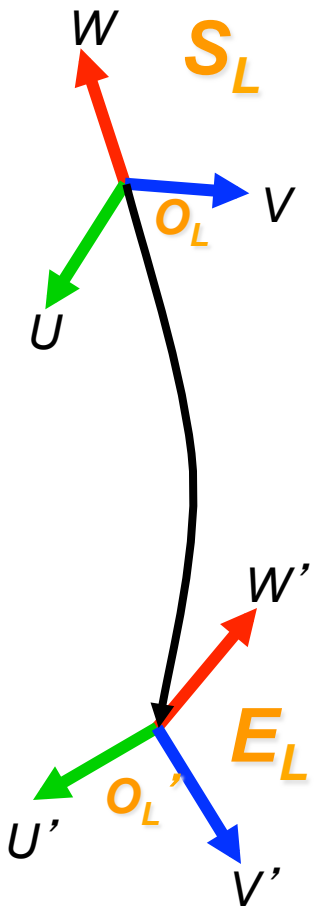
2D: polytwist



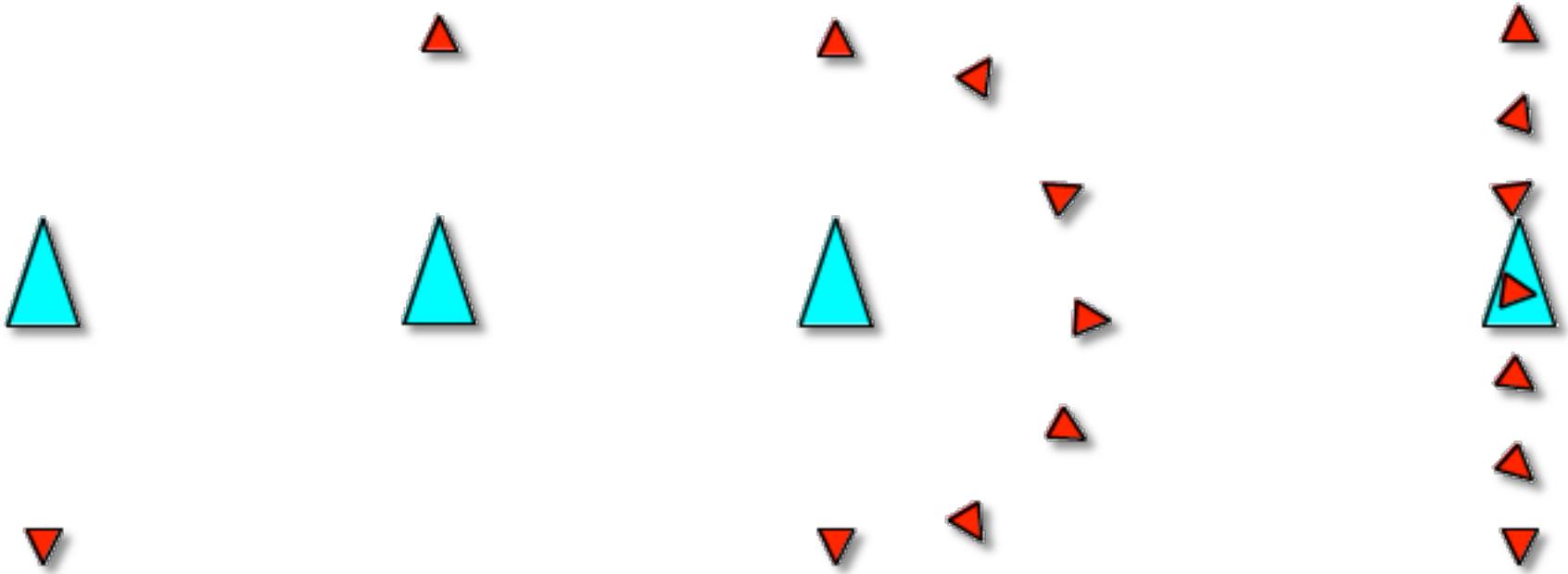




3D: polyscrew



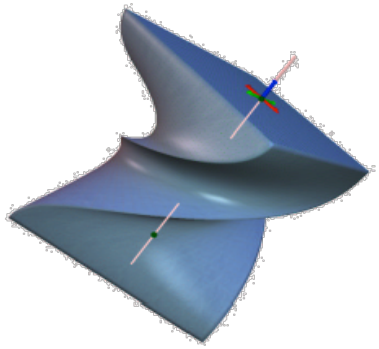
Motion independent of CS



Screw history

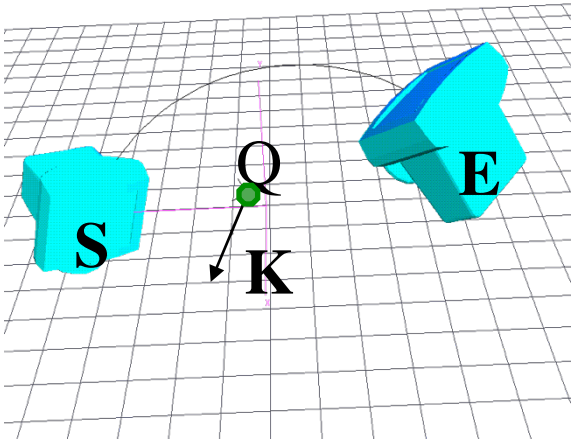
(Ceccarelli [2000] Detailed study of screw motion history)

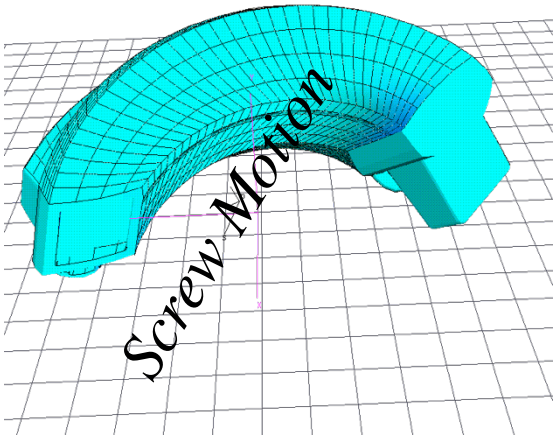
- Archimede (287–212 BC) designed helicoidal screw for water pumps
- Leonardo da Vinci (1452–1519) description of helicoidal motion
- Dal Monte (1545–1607) and Galileo (1564–1642) mechanical studies on helicoidal geometry
- Giulio Mozzi (1763) screw axis as the "spontaneous axis of rotation"
- L.B. Francoeur (1807) theorem of helicoidal motion
- Gaetano Giorgini (1830) analytical demonstration of the existence of the "axis of motion" (locus of minimum displacement points)
- Ball (1900) "*Theory of screws*"
- Rodrigues (1940) helicoidal motion as general motion
- ….
- Zefrant and Kumar (CAD 1998) Interpolating motions



Why screws?

- Screw motions are great!
  - Uniquely defined by start pose S and end pose E
  - Independent of coordinate system
  - Subsumes pure rotations and translations
  - Minimizes rotation angle & translation distance
  - Natural motions for many application
- Simple to apply for any value of t in [0,1]
  - Rotation by angle ***tb*** around axis Axis(**Q**,**K**)
  - Translation by distance ***td*** along Axis(**Q**,**K**)
  - Each point moves along a **helix**
- Simple to compute from poses S and E
  - Axis: point **Q** and direction **K**
  - Angle **b**
  - Distance **d**





Computing the screw parameters

From initial and final poses:

M(0) and M(1)

K:=(U' –U)×(V' –V);

K:=K / ||K||;

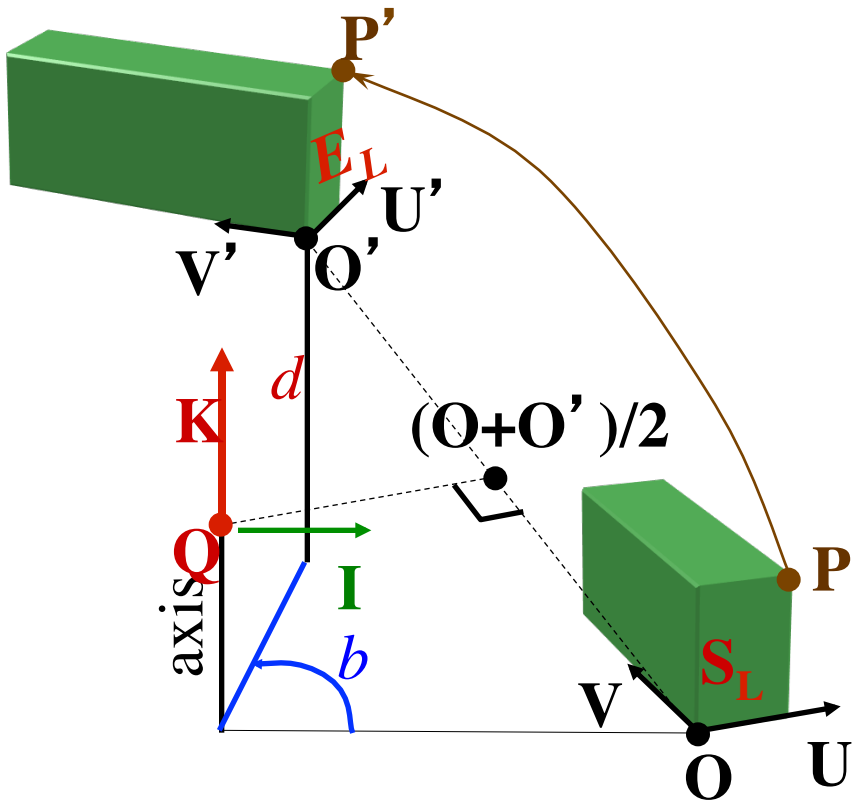
$b := 2 \sin^{-1}(\|U' - U\| / (2 \|K \times U\|))$;

d:=K•OO' ;

Q:=(O+O')/2 + (K×OO') / (2tan(b/2));

To apply a t-fraction of the screw:

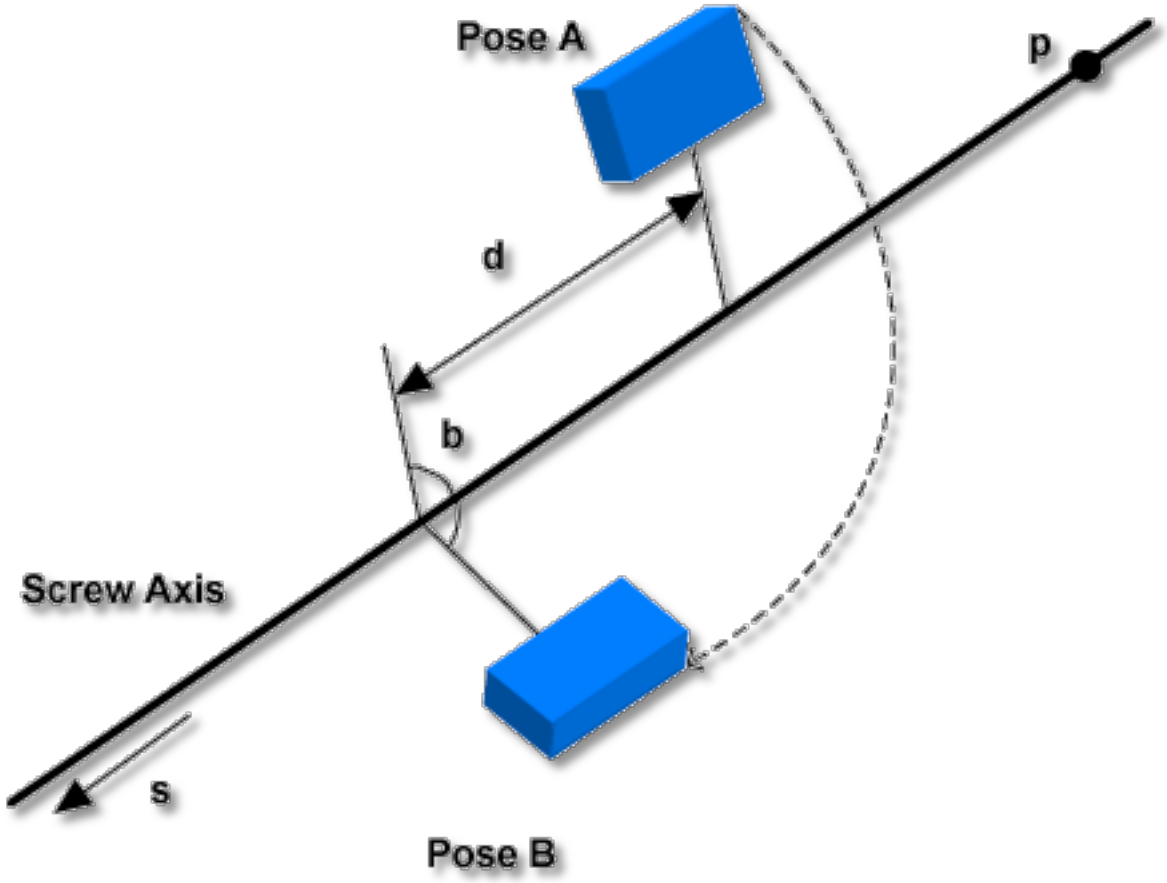
- Translate by –Q;
- Rotate around K by tb;
- Translate by (0,0,td);
- Translate by Q;



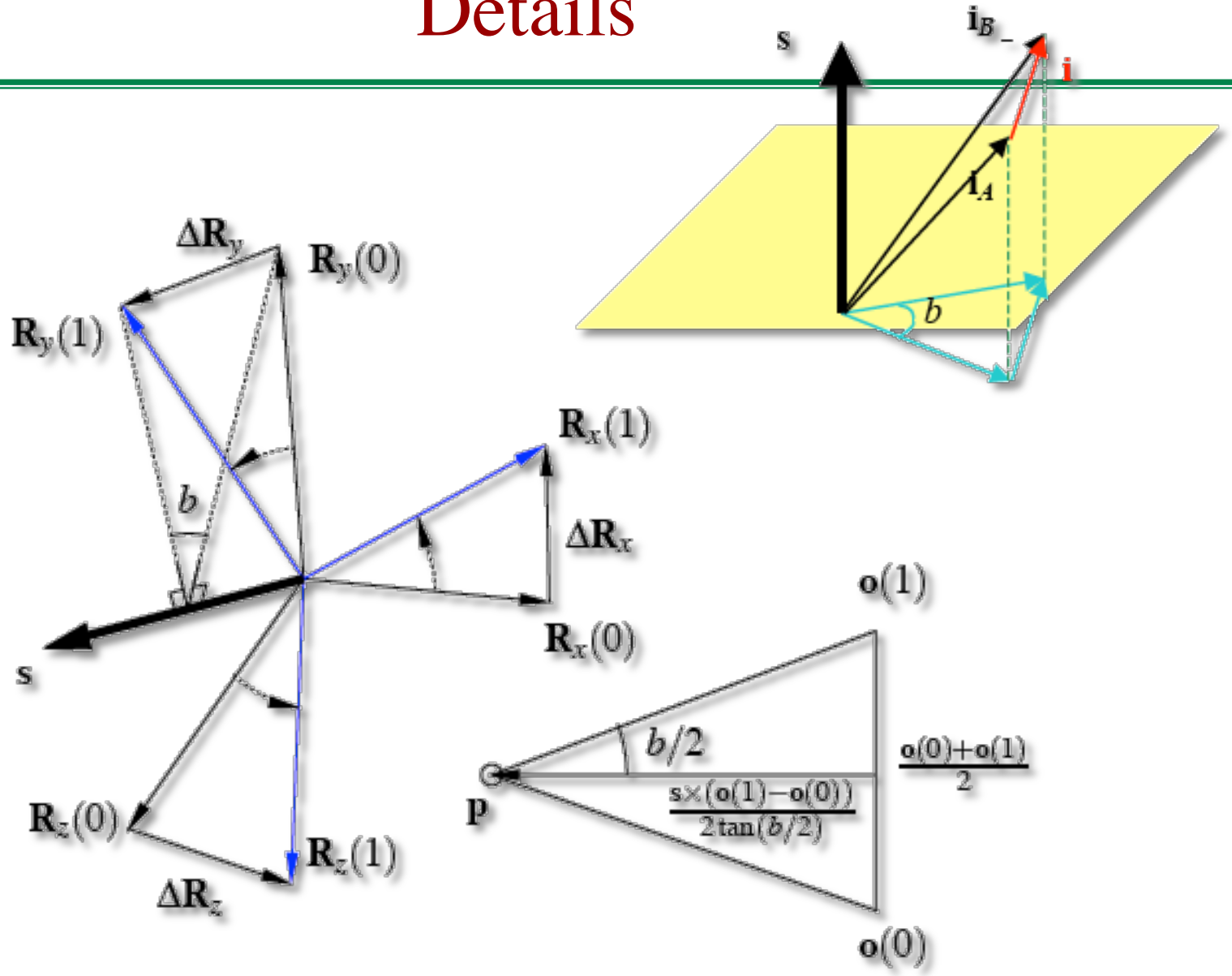
Details

s=K

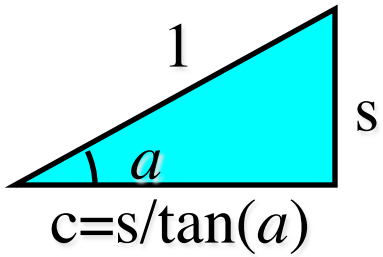
p=Q

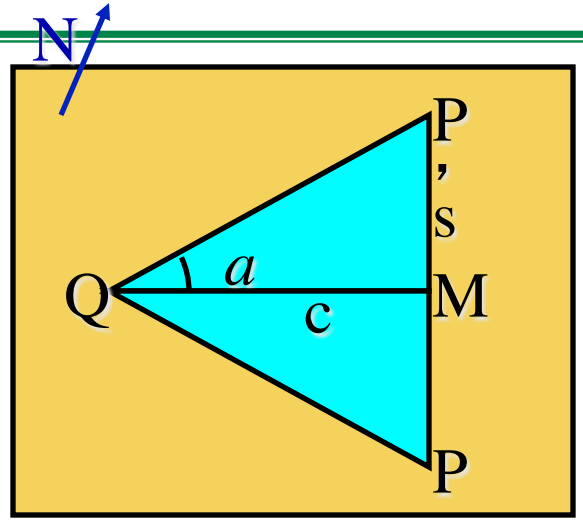


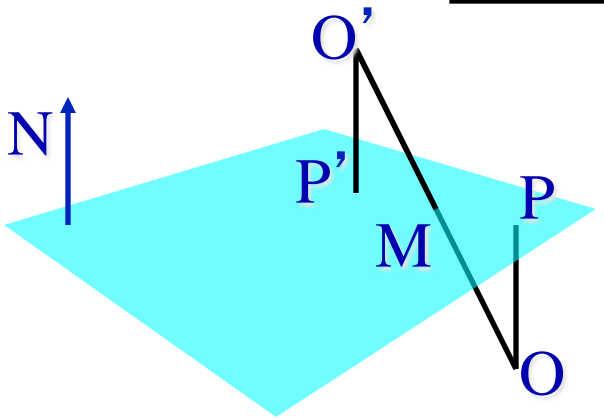
Details


s
i_B
i
i_A
b
ΔR_y
$R_y(0)$
$R_y(1)$
b
$R_x(1)$
ΔR_x
$R_x(0)$
s
$R_z(0)$
$R_z(1)$
ΔR_z
o(1)
b/2
$\frac{o(0)+o(1)}{2}$
p
$\frac{s\times(o(1)-o(0))}{2\tan(b/2)}$
o(0)

Computing point Q on screw axis







Given P, P' , and *a*, compute Q the center of rotation by angle *b*=2*a* that brings P to P'

M=(P+P')/2

T=N×PP' /||PP' ||

c=||PP' ||/(2tan(a)

A=M+cT

=(P+P')/2+N×PP' /(2tan(a)

P=O+hN, P' =O' –hN (projections)

P+P' =O+O' , PP' =OO' –2hN

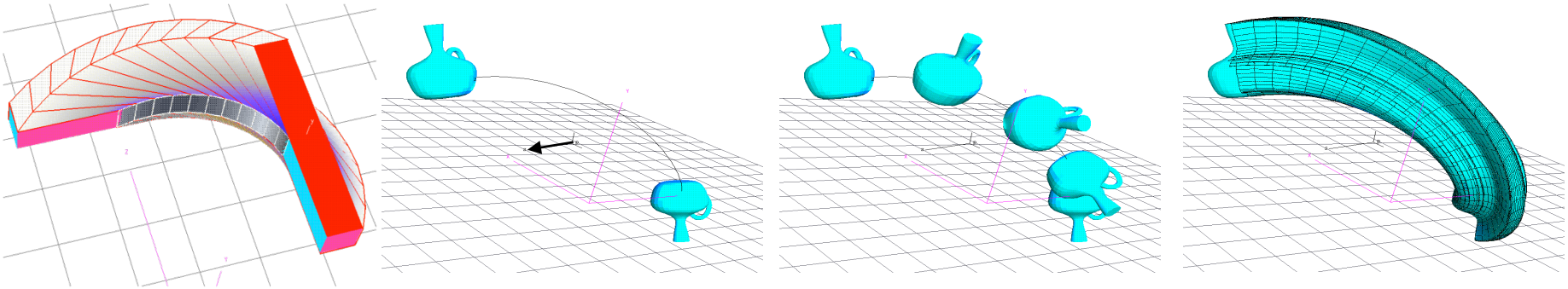
N×PP' = N×OO' – 2hN×N = N×OO'

A = (O+O')/2 + N×OO' /(2tan(*b*/2)

Volume swept during screw motion

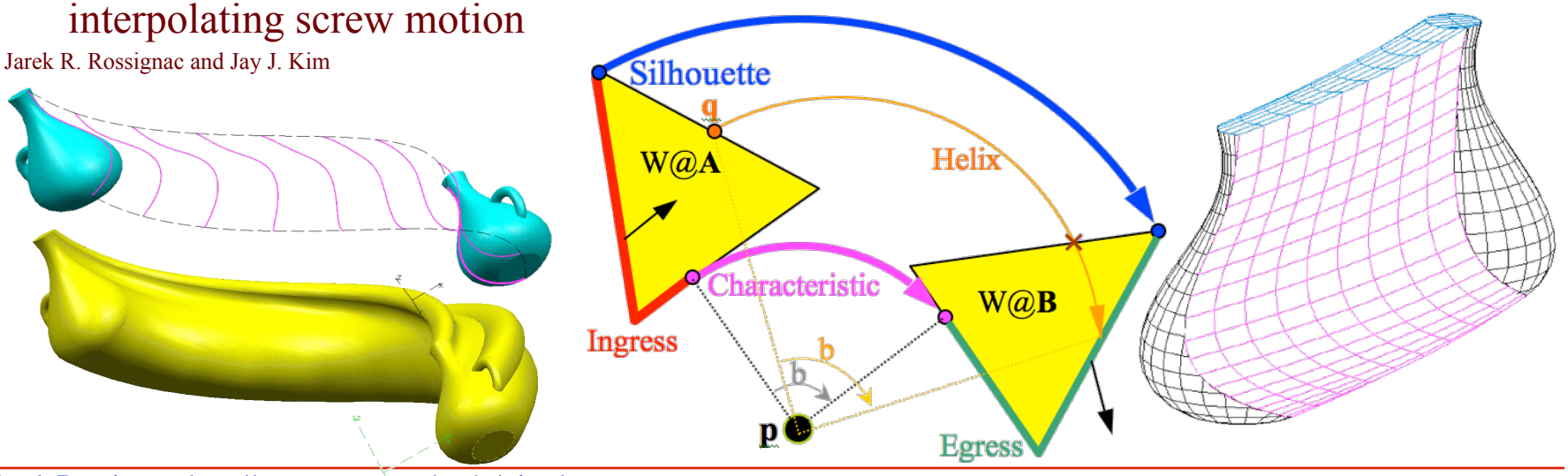
Computing and visualizing pose-interpolating 3D motions

Jarek R. Rossignac and Jay J. Kim (Hanyang University, Seoul, Korea), CAD, 33(4)279:291, April 2001.



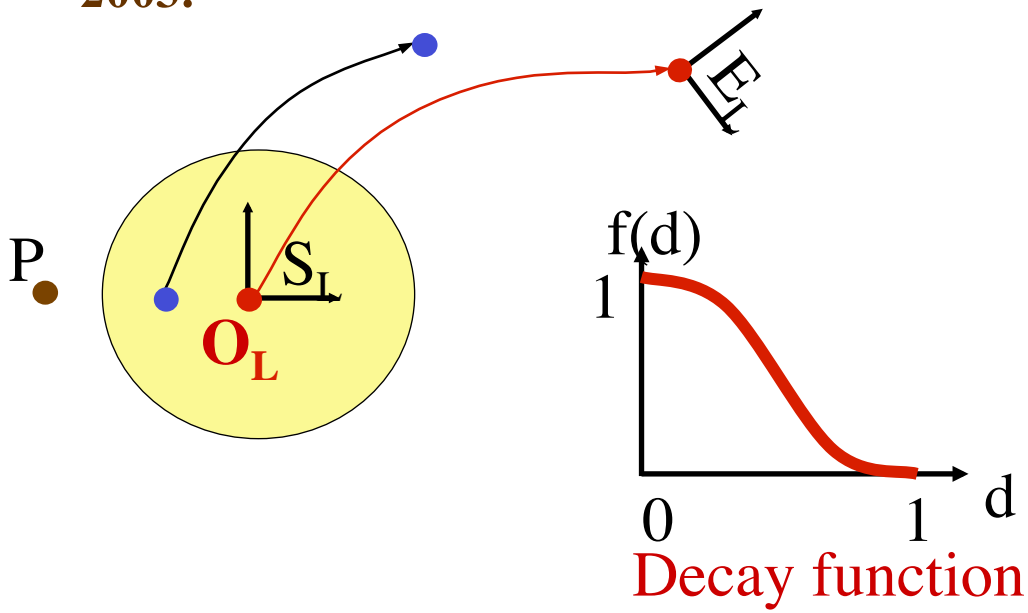
SweepTrimmer: Boundaries of regions swept by sculptured solids during a pose-interpolating screw motion

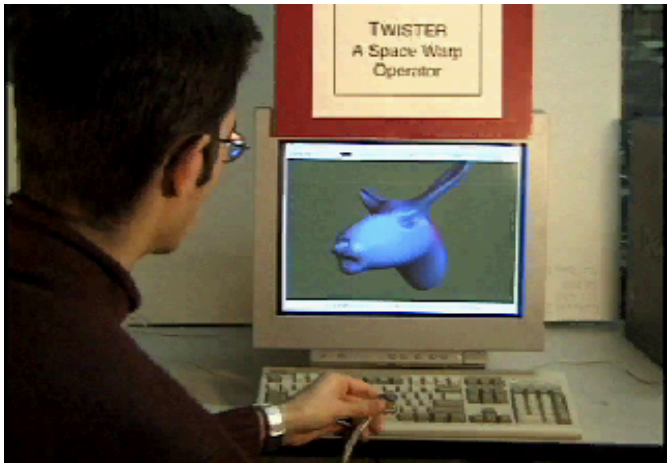
Jarek R. Rossignac and Jay J. Kim

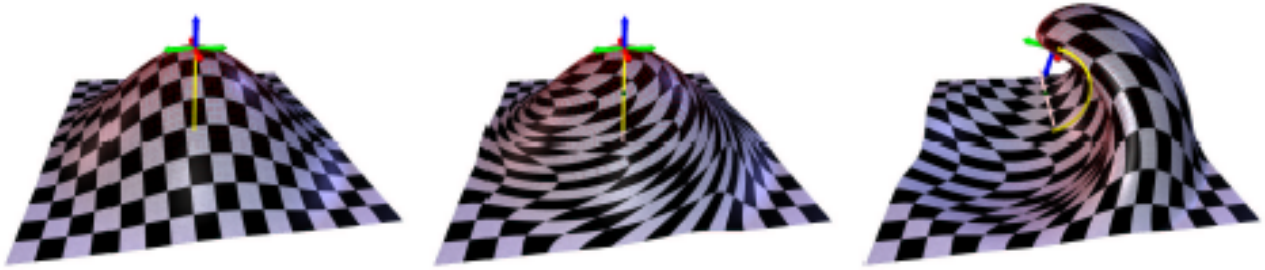


Space warp based on a screw motion

"Twister: A space-warp operator for the two-handed editing of 3D shapes", Llamas, Kim, Gargus, Rossignac, and Shaw. Proc. ACM SIGGRAPH, **July 2003.**

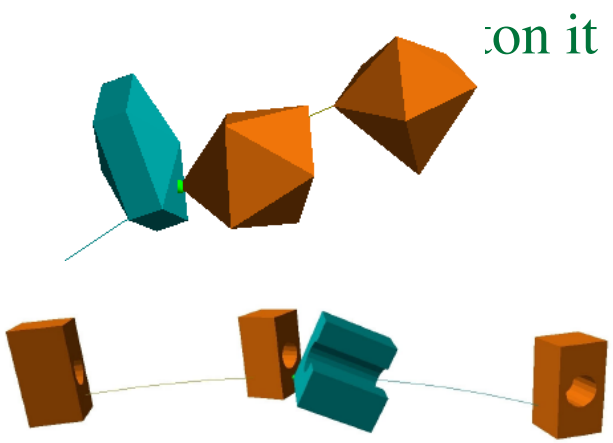


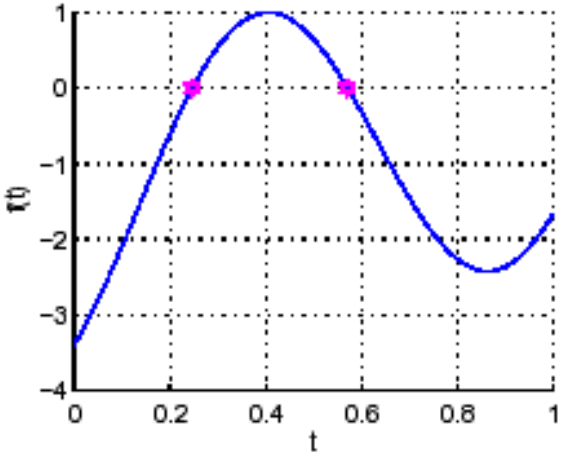


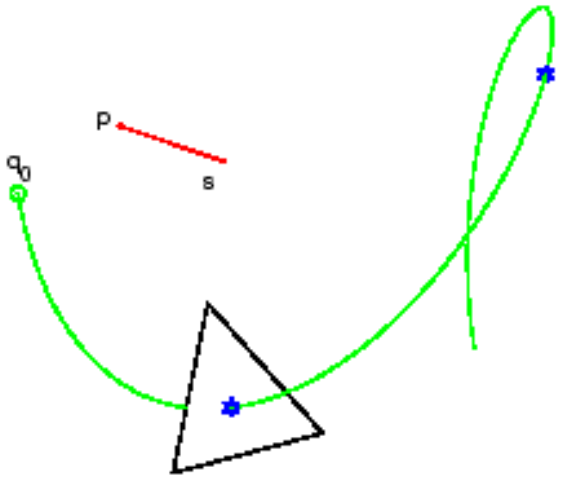


Vertex-face (helix-plane intersection)

- Helix is $\mathbf{V}(t) = r\cos(tb)\mathbf{i}+r\sin(tb)\mathbf{j}+td\mathbf{k}$ in screw coordinates
  - where V(0) lies on the **i**-axis and the **k** is the screw axis
- **The screw intersects plane d +V(t)•n = 0 for values of t satisfying**
  - $d+(r\cos(tb), r\sin(tb), td)\bullet n = 0$
- **We compute all roots and check if they correspond to points in triangle**
  - Reduces to finding roots of f(t)=A+Bt+Ccos(bt+c)
  - Separate roots using f' (t)=0, which requires solving B/bC=sin(bt +c)

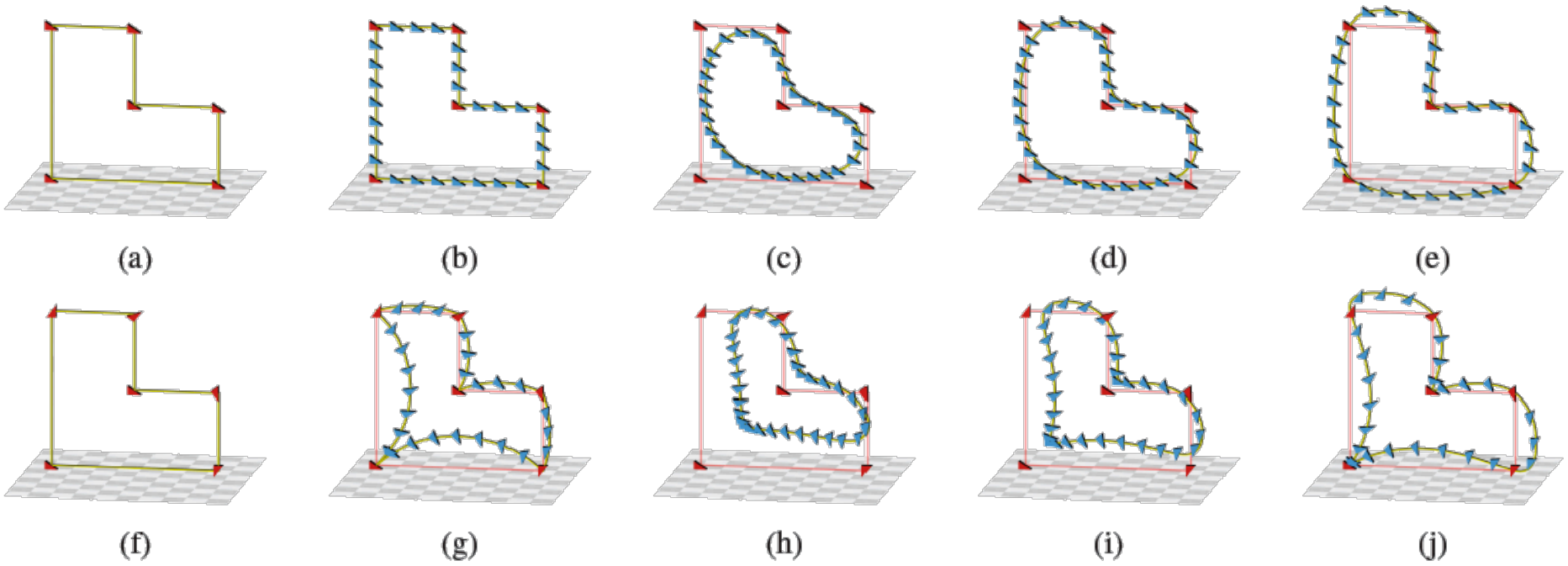




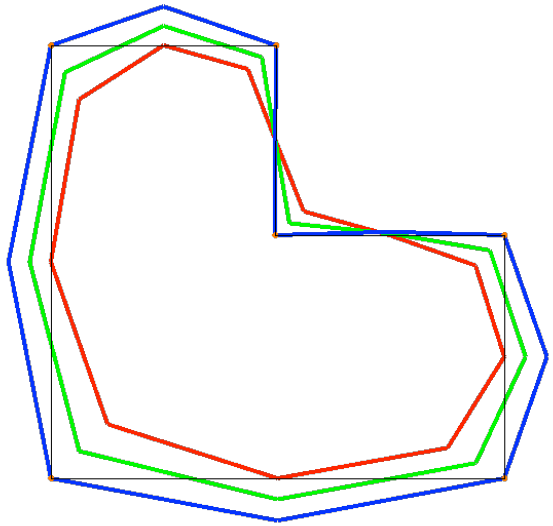


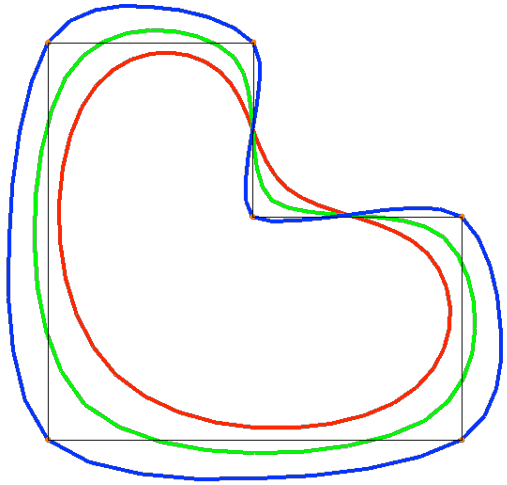
Polyscrew smoothing

Parallel control poses (a) define a piecewise linear animation with sharp changes of velocity (b).
The polyscrew may be smoothed by a cubic B-spline (c), Jarek (d), or 4-point subdivision (e).
Rotated control poses (f) define a polyscrew (g) interpolating consecutive poses.
It may be smoothed (ScrewBender) via B-spline (h), Jarek (i) and 4-point (j) subdivisions.
The three subdivided motions are at least C1. The B-spline is C2.

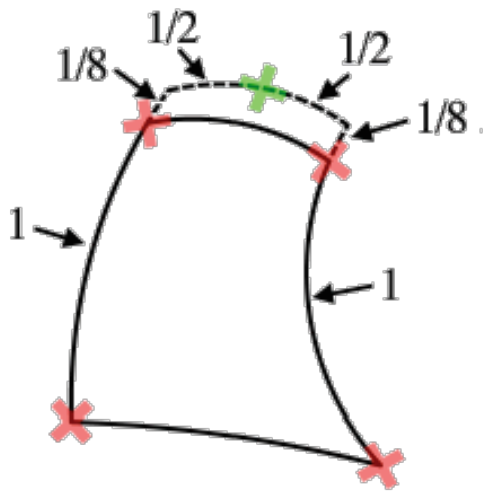


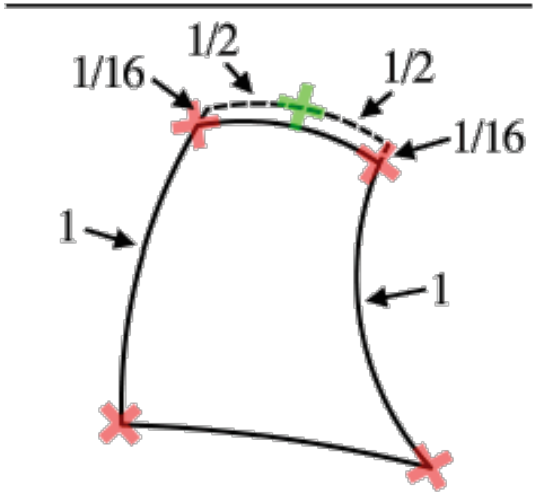
Details of polyscrew subdivision



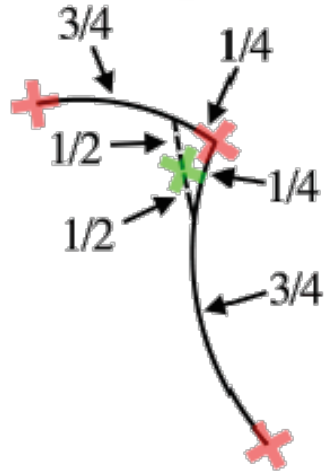


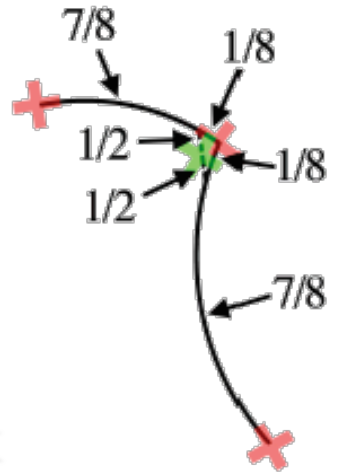
4-point





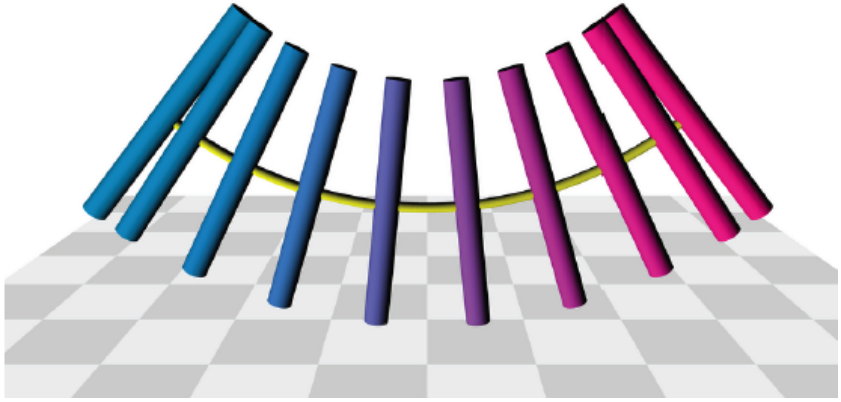
B-spline



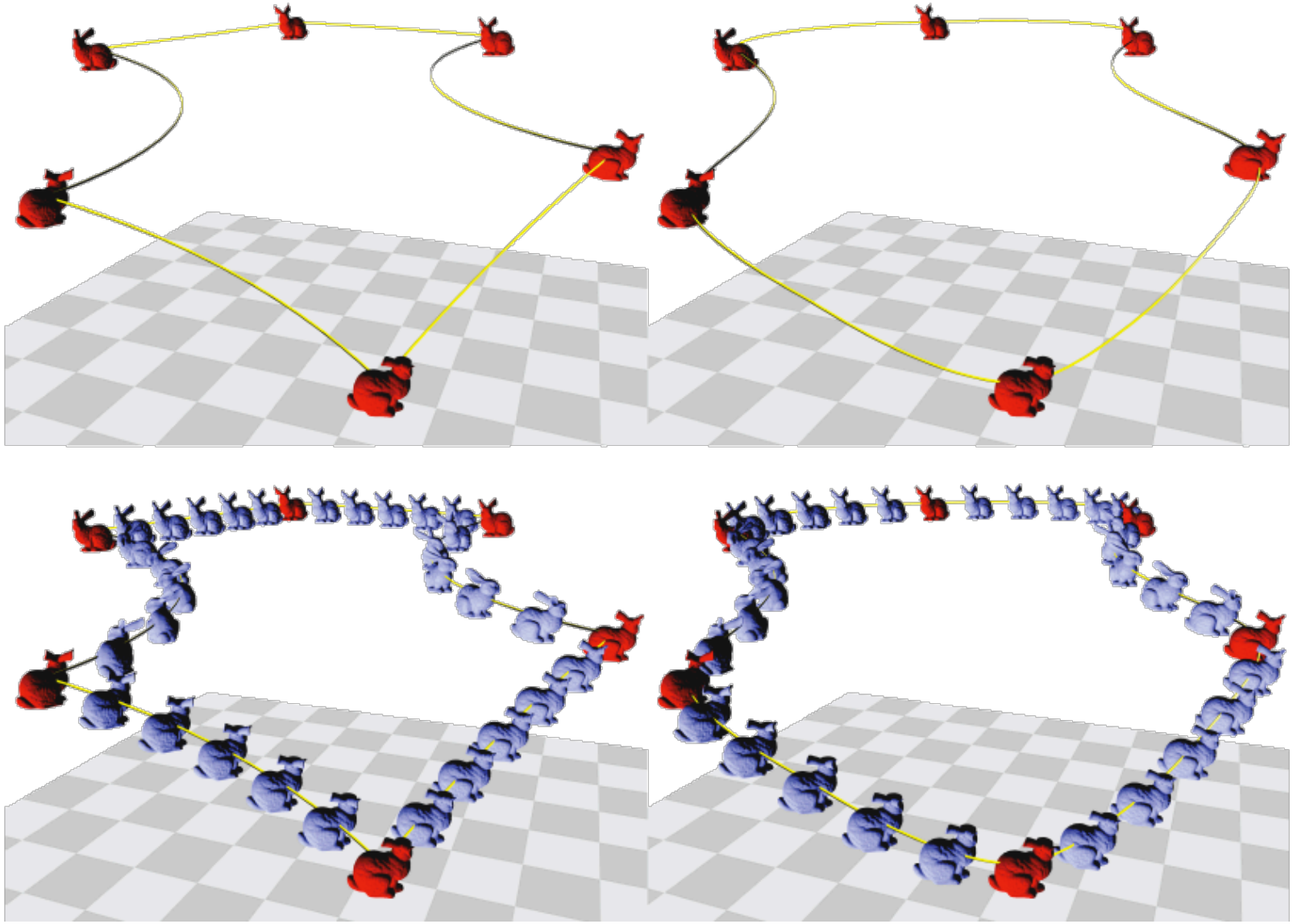


jarek

Example with only 2 control poses



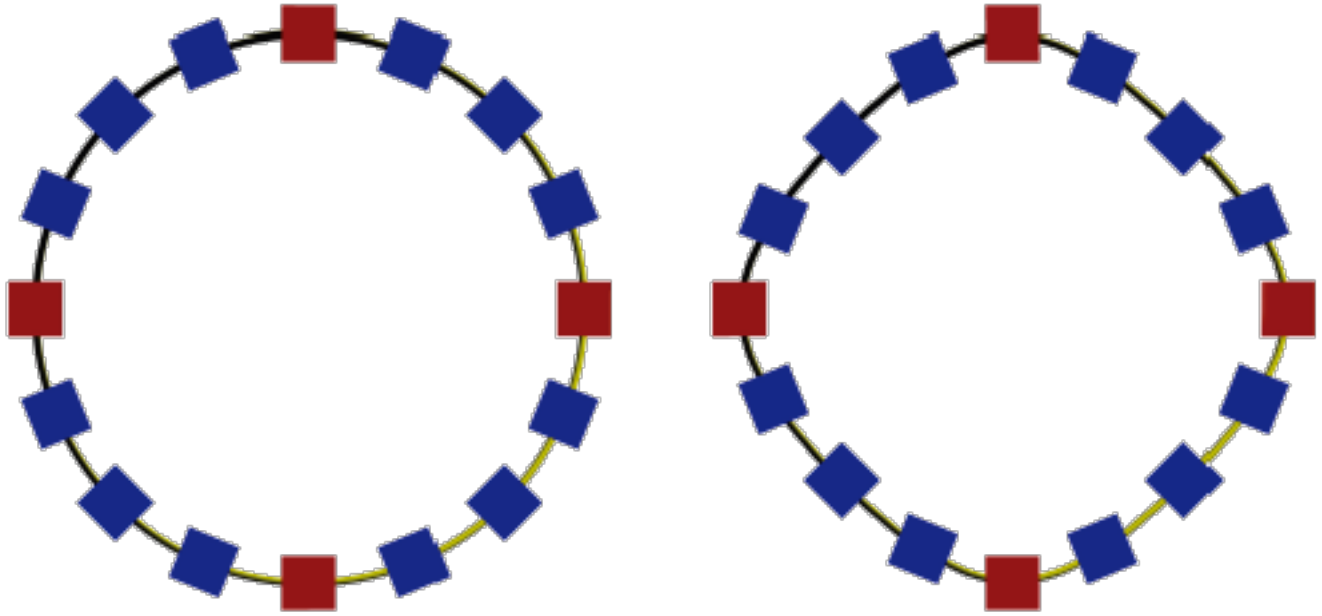
Flying bunny example



Comparison

ScrewBender

4-point trajectory
and linear rotation



(a) (b)

Poses in processing

```
class pose {
  pt O = new pt(0,0,0);
  vec U = new vec(0,1,0);
  vec I = new vec(1,0,0);
  vec J = new vec(0,1,0);
  vec K = new vec(0,0,1);
  pose(vec T, vec U) {        // create pose from K and up-vector
     vec UxT=cross(U,T); float n=UxT.norm();
     if (n<0.0001) {…} else {
     K.setTo(T); K.makeUnit();
     I.setTo(UxT); I.makeUnit();
     J.setTo(cross(K,I)); };
    };
```

Rotating a pose

```
void Rx(float a) {
   float c=cos(a), s=sin(a);
   vec cJ=J.make(); cJ.mul(c);
   vec msJ=J.make(); msJ.mul(-s);
   vec sK=K.make(); sK.mul(s);
   vec cK=K.make(); cK.mul(c);
   J.setTo(sum(cJ,sK));
   K.setTo(sum(msJ,cK));
   }
```

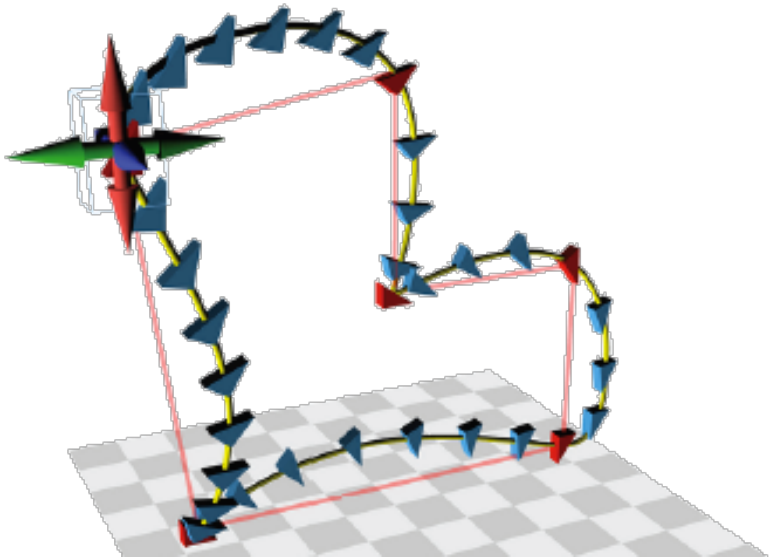
Transforming points and vectors

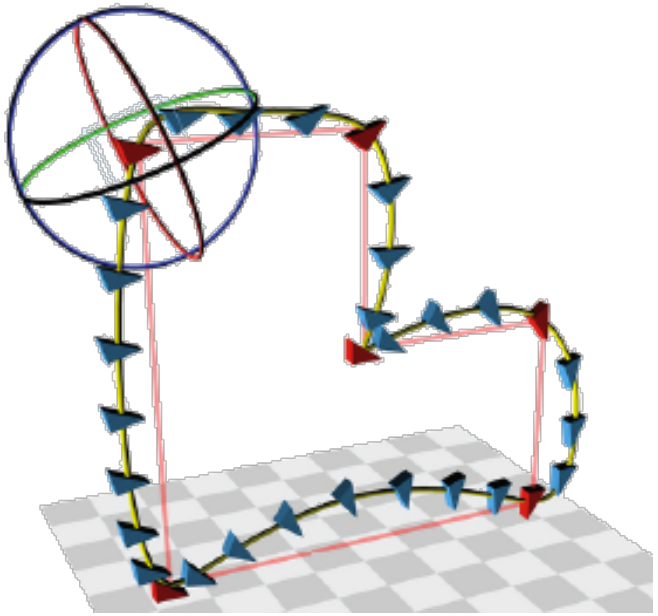
```
vec transform(vec V) {
    vec R=new vec(0,0,0);
    R.addScaled(V.x,I);
    R.addScaled(V.y,J);
    R.addScaled(V.z,K);
    return(R); };

 pt transform(pt P) {
    pt R=O.make();
    R.addScaledVec(P.x,I);
    R.addScaledVec(P.y,J);
    R.addScaledVec(P.z,K);
    return(R); };
```

Screw a pose towards another

```
void screw(pose P0, float s, pose P1) {
  pt O0=P0.O;   pt O1=P1.O; vec dO=O0.vecTo(O1);
  vec I0=P0.I;   vec I1=P1.I; vec dI=dif(I1,I0);
  vec J0=P0.J;   vec J1=P1.J; vec dJ=dif(J1,J0);
  vec K0=P0.K;   vec K1=P1.K; vec dK=dif(K1,K0);
   vec T=cross(dK,dI); T.add(cross(dI,dJ)); T.add(cross(dJ,dK));
   T.makeUnit();
   vec Ni=cross(T,I0); float ni=Ni.norm();  vec Nj=cross(T,J0); float nj=Nj.norm();  vec Nk=cross(T,K0); float nk=Nk.norm();
     vec N0=new vec(0,0,0), N1=new vec(0,0,0);
     N0=Ni; N1=cross(T,I1); if (nj>ni) {N0=Nj; N1=cross(T,J1);}; if ((nk>ni)&&(nk>nj)) {N0=Nk; N1=cross(T,K1);};
    pose S =  new pose(T,N0);
     vec nN1=S.inverse(N1);
     float b = atan2(-nN1.x,nN1.y);
     float d = dot(T,dO);
    pt P = midPt(O0,O1); vec H = cross(T,dO); H.mul(1.0/tan(b/2)/2); P.addVec(H);
        screwAxisDirection.setTo(T); screwAxisDirection.mul(10*b/PI); screwAxisPoint.setTo(P);
     S.O.setTo(P);
   pt nO = S.inverse(O);  vec nI = S.inverse(I); vec nJ = S.inverse(J); vec nK = S.inverse(K);
  S.Rz(s*b); S.Tz(s*d);
  O.setTo(S.transform(nO));  I.setTo(S.transform(nI));  J.setTo(S.transform(nJ)); K.setTo(S.transform(nK));
 };  }
```

Interactive motion editing





Quaternions

Ken Shoemake Siggraph '85 (Computer Graphics, V. 19, No. 3, P.245)

Same information as axis/angle but in a different form

$$q = Rot_{\theta,(x,y,z)} = [\cos(\theta/2), \sin(\theta/2)\cdot(x,y,z)]$$

$$v' = Rot_q(v) = q \cdot v \cdot q^{-1}$$

$$[s_1, v_1] + [s_2, v_2] = [s_1 + s_2, v_1 + v_2]$$

$$[s_1, v_1] \bullet [s_2, v_2] = [s_1 \cdot s_2 - v_1 \bullet v_2, s_1 \bullet v_2 + s_2 \bullet v_1 + v_1 \times v_2]$$

$$slerp(q1, q2, u)$$
$$= ((\sin((1-u)\cdot\theta))/(\sin\theta))\cdot q_1 + (\sin(u\cdot\theta))/(\sin\theta)\cdot q_2$$