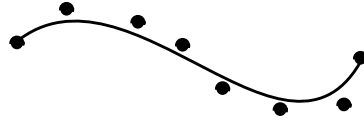
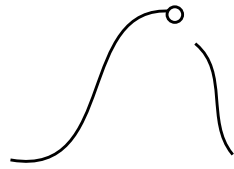


a) an interpolating spline in which the spline passes through the interior control points

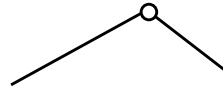


b) an approximating spline in which only the end points are interpolated; the interior control points are only used to design the curve

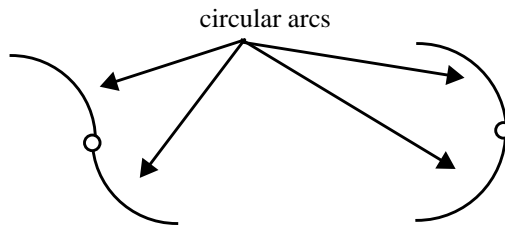
FIGURE 28. Comparing interpolation and approximating splines.



a) positional discontinuity at the point



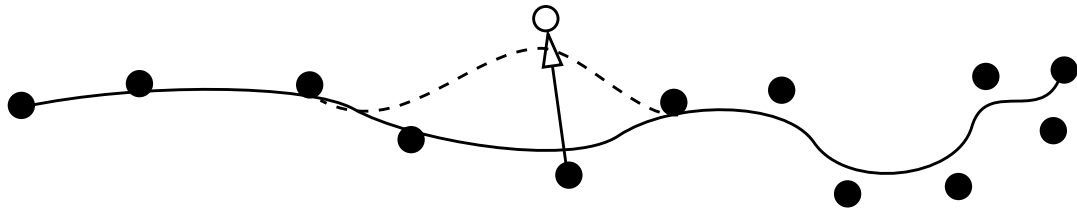
b) positional continuity but not tangential continuity at the point



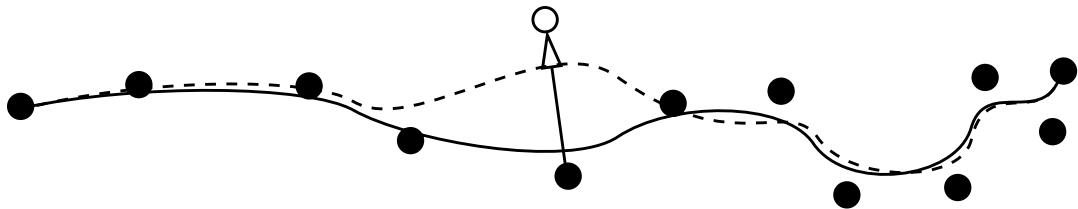
c) positional and tangential continuity but not curvature continuity at the point

d) positional, tangential, and curvature continuity at the point

FIGURE 29. Continuity (at the point indicated by the small circle).



a) local control: moving one control point only changes the curve over a finite bounded region



b) global control: moving one control point changes the entire curve;
distant sections may change only slightly.

FIGURE 30. Comparing local and global effect of moving a control point

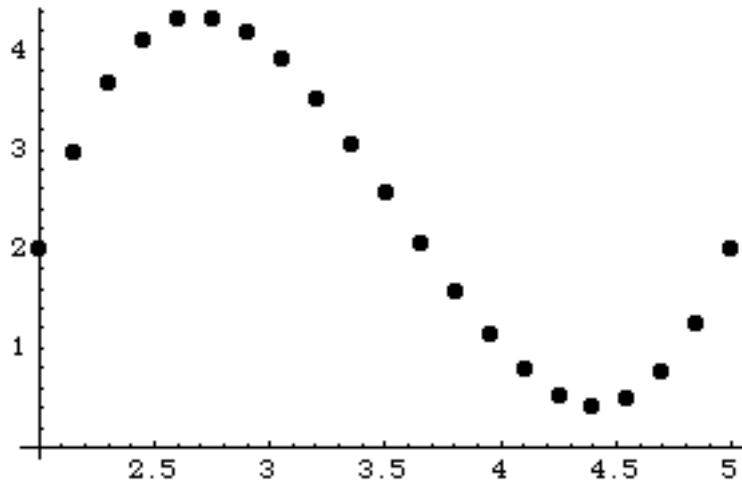


FIGURE 31. Example of points produced by equal increments of an interpolating parameter for a typical cubic curve.

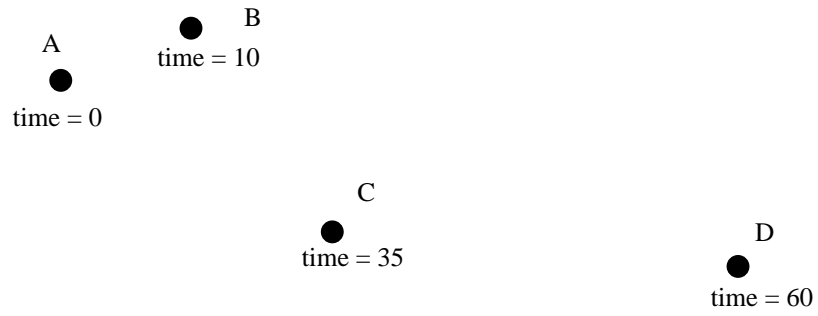


FIGURE 32. Position-time pairs constraining the motion.

$$P(u) = au^3 + bu^2 + cu + d \quad (\text{EQ 29})$$

$$P(u) = (x(u), y(u), z(u)) \quad (\text{EQ 30})$$

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$

$$z(u) = a_z u^3 + b_z u^2 + c_z u + d_z$$

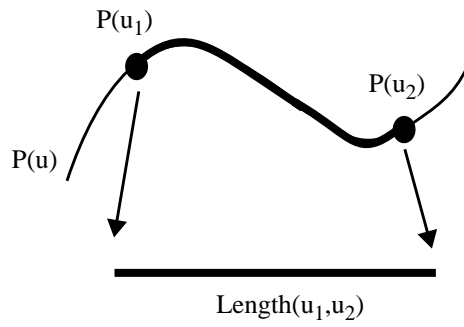


FIGURE 33. $\text{Length}(u_1, u_2)$.

$$s = \int_{u1}^{u2} |dP/du| du \quad (\text{EQ 31})$$

$$dP/du = ((dx(u)/(du)),(dy(u)/(du)),(dz(u)/(du))) \quad (\text{EQ 32})$$

$$|dP/du| = \sqrt{(dx(u)/du)^2 + (dy(u)/du)^2 + (dz(u)/du)^2} \quad (\text{EQ 33})$$

$$A \cdot u^4 + B \cdot u^3 + C \cdot u^2 + D \cdot u + E \quad (\text{EQ 34})$$

$$\begin{aligned} A &= 9 \cdot (a_x^2 + a_y^2) \\ B &= 12 \cdot (a_x \cdot b_x + a_y \cdot b_y) \\ C &= 6 \cdot (a_x \cdot c_x + a_y \cdot c_y) + 4 \cdot (b_x^2 + b_y^2) \\ D &= 4 \cdot (b_x \cdot c_x + b_y \cdot c_y) \\ E &= c_x^2 + c_y^2 \end{aligned} \quad (\text{EQ 35})$$

Index	Parametric Entry	Arc Length
0	0.00	0.000
1	0.05	0.080
2	0.10	0.150
3	0.15	0.230
4	0.20	0.320
5	0.25	0.400
6	0.30	0.500
7	0.35	0.600
8	0.40	0.720
9	0.45	0.800
10	0.50	0.860
11	0.55	0.900
12	0.60	0.920
13	0.65	0.932
14	0.70	0.944
15	0.75	0.959
16	0.80	0.972
17	0.85	0.984
18	0.90	0.994
19	0.95	0.998
20	1.00	1.000

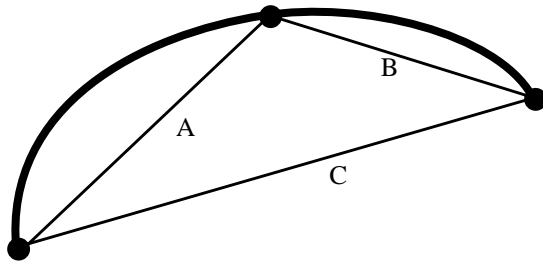
FIGURE 34. Table of parameter, arc length pairs.

$$i = (int)\left(\frac{\text{given parametric value}}{\text{distance between entries}} + 0.5\right) = (int)\left(\frac{0.73}{0.05} + 0.5\right) = 15 \quad \text{(EQ 36)}$$

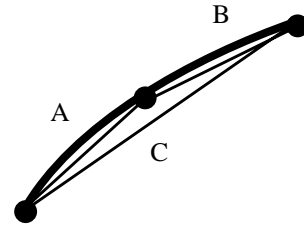
$$i = (int)\left(\frac{\text{given parametric value}}{\text{distance between entries}}\right) = (int)\left(\frac{0.73}{0.05}\right) = 14 \quad \text{(EQ 37)}$$

$$\begin{aligned} L &= \text{ArcLength}[i] + \frac{(\text{GivenValue} - \text{Value}[i])}{(\text{Value}[i+1] - \text{Value}[i])} \cdot (\text{ArcLength}[i+1] - \text{ArcLength}[i]) \\ &= 0.944 + \frac{0.73 - 0.70}{0.75 - 0.70} \cdot (0.959 - 0.944) \\ &= 0.953 \end{aligned} \tag{EQ 38}$$

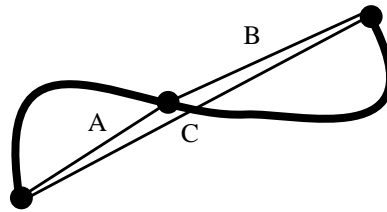
$$4.0 + \left(\frac{3}{8}\right) \cdot (4.5 - 4.0) = 4.1875 \tag{EQ 39}$$



Lengths $A+B-C$ above error tolerance



Lengths $A+B-C$ within error tolerance



Lengths $A+B-C$ erroneously reports that the error is within tolerance

FIGURE 35. Tests for adaptive subdivision

$$\int_{-1}^1 f(u) = \sum_i w_i f(u_i) \quad (\text{EQ 40})$$

$$u = (2 \cdot t - a - b)/(b - a) \quad (\text{EQ 41})$$

$$\int_a^b g(t) = \int_{-1}^1 f(u) = \int_{-1}^1 ((b-a)/2 \cdot g(((b-a) \cdot u + b + a)/2)) \quad (\text{EQ 42})$$

$$\int_{-1}^1 \sqrt{A \cdot u^4 + B \cdot u^3 + C \cdot u^2 + D \cdot u + E} \quad (\text{EQ 43})$$

```

/* -----
STRUCTURES
*/

// the structure to hold entries of the table consisting of
// parameter value (u) and estimated length (length)
typedef struct table_entry_structure {
    double u,length;
} table_entry_td;

// the structure to hold an interval of the curve, defined by
// starting and ending parameter values and the estimated
// length (to be filled in by the adaptive integration
// procedure
typedef struct interval_structure {
    double u1,u2;
    double length;
} interval_td;

// coefficients for a 2D cubic curve
struct cubic_curve_structure {
    double ax,bx,cx,dx;
    double ay,by,cy,dy;
} typedef cubic_curve_td;

// polynomial function structure; a quadric function is generated
// from a cubic curve during the arclength computation
struct quadric_poly_structure {
    double *coeff;
    int degree;
} typedef polynomial_td;

/* -----
ADAPTIVE INTEGRATION
this is the high level call used whenever a curve's length is to be computed
*/
void adaptive_integration(cubic_curve_td *curve, double u1, double u2, double tolerance)
{
    double subdivide();
    polynomial_tdfunc;
    interval_td full_interval;
    double total_length;
    double integrate_func();
    double temp;

    func.degree = 4;
    func.coeff = (double *)malloc(sizeof(double)*5);
    func.coeff[4] = 9*(curve->ax*curve->ax + curve->ay*curve->ay);
    func.coeff[3] = 12*(curve->ax*curve->bx + curve->ay*curve->by);
    func.coeff[2] = (6*(curve->ax*curve->cx + curve->ay*curve->cy) +
        4*(curve->bx*curve->bx + curve->by*curve->by) );
    func.coeff[1] = 4*(curve->bx*curve->cx + curve->by*curve->cy);

```

```

func.coeff[0] = curve->cx*curve->cx + curve->cy*curve->cy;

full_interval.u1 = u1; full_interval.u2 = u2;
temp = integrate_func(&func,&full_interval);
printf("\nInitial guess = %lf; %lf:%lf",temp,u1,u2);
full_interval.length = temp;
total_length = subdivide(&full_interval,&func,0.0,tolerance);
printf("\n total length = %lf\n",total_length);
}

/* -----
SUBDIVIDE
'total_length' is the length of the curve up to, but not including, the 'full_interval'
if the difference between the interval and the sum of its halves is less than 'tolerance',
    stop the recursive subdivision
'func' is a polynomial function
*/
double subdivide(interval_td *full_interval, polynomial_td *func,
                double total_length, double tolerance)
{

    interval_topleft_interval, right_interval;
    double left_length,right_length;
    double midu;
    double subdivide();
    double integrate_func();
    double temp;
    void add_table_entry();

    midu = (full_interval->u1+full_interval->u2)/2;
    left_interval.u1 = full_interval->u1; left_interval.u2 = midu;
    right_interval.u1 = midu; right_interval.u2 = full_interval->u2;

    left_length = integrate_func(func, left_interval);
    right_length = integrate_func(func,right_interval);

    temp = fabs(full_interval->length - (left_length+right_length));
    if (temp > tolerance) {
        left_interval.length = left_length;
        right_interval.length = right_length;
        total_length = subdivide(&left_interval, func, total_length, tolerance);
        total_length = subdivide(&right_interval, func, total_length, tolerance);
        return(total_length);
    }
    else {
        total_length = total_length + left_length;
        add_table_entry(midu,total_length);
        total_length = total_length + right_length;
        add_table_entry(full_interval->u2,total_length);
        return(total_length);
    }
}
}

```

```

/* -----
ADD TABLE ENTRY
adds an entry of the form (parametric value, length) to the table being constructed
*/
void add_table_entry(double u, double length)
{
    /* add entry of (u, length) */
    printf("\ntable entry: u: %lf, length: %lf",u,length);
}

/* -----
INTEGRATE FUNCTION
use gaussian quadrature to integrate square root of given function in the given interval
*/
double integrate_func(polynomial_td *func,interval_td *interval)
{
    double x[5]={.1488743389,.4333953941,.6794095682,.8650633666,.9739065285};
    double w[5]={.2966242247,.2692667193,.2190863625,.1494513491,.0666713443};
    double length, midu, dx, diff;
    int i;
    double evaluate_polynomial();
    double u1,u2;

    u1 = interval->u1;
    u2 = interval->u2;

    midu = (u1+u2)/2.0;
    diff = (u2-u1)/2.0;
    length = 0.0;
    for (i=0; i<5; i++) {
        dx = diff*x[i];
        length += w[i]*(sqrt(evaluate_polynomial(func,midu+dx)) +
            sqrt(evaluate_polynomial(func,midu-dx)));
    }
    length *= diff;

    return (length);
}

/* -----
EVALUATE POLYNOMIAL
evaluate a polynomial
*/
double evaluate_polynomial(polynomial_td *poly, double u)
{
    double w;
    int i;
    double value;

    value = 0.0;
    w = 1.0;
    for (i=0; i<=poly->degree; i++) {
        value += poly->coeff[i]*w;
    }
}

```

```
    w *= u;
  }
  return value;
}
```

FIGURE 36. Adaptive Gaussian integration of arc length.

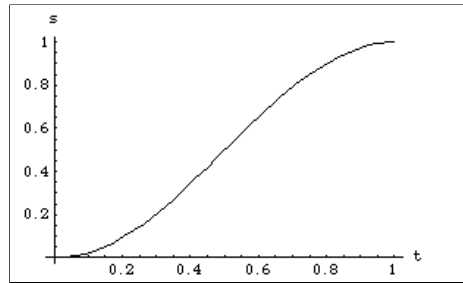


FIGURE 37. Ease(t).

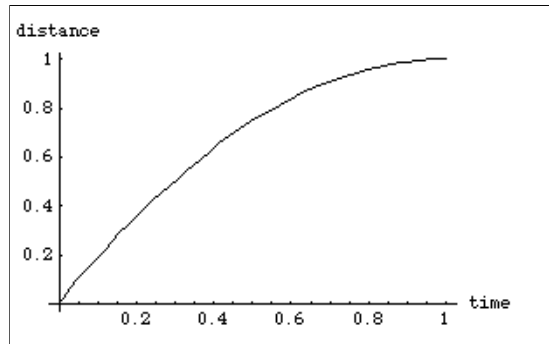
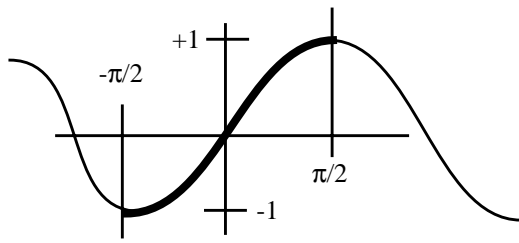
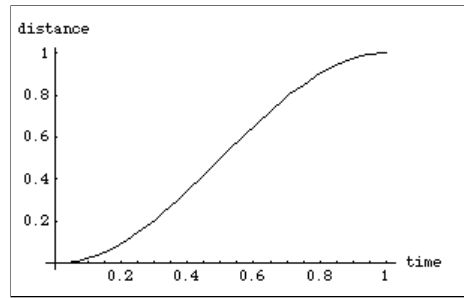


FIGURE 38. Graph of sample analytic distance-time function $(2-t)*t$

$$s(t) = \text{ease}(t) = \frac{\sin t \cdot \pi - \frac{\pi}{2} + 1}{2} \quad (\text{EQ 44})$$



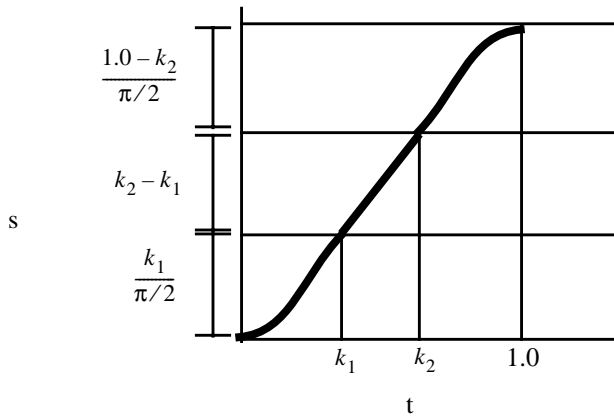
a) Sine curve segment to use as ease-in/ease-out control



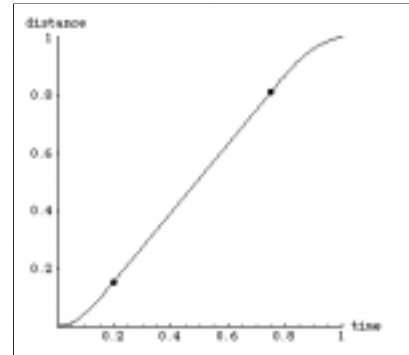
b) Sine curve segment mapped to useful values

FIGURE 39. Using a sine curve segment as the ease-in/ease-out distance-time function

$$\frac{k_1}{\pi/2} + k_2 - k_1 + \frac{1.0 - k_2}{\pi/2} .$$



a) Ease-in/ease-out curve as it is initially pieced together



b) curve segments scaled into useful values with points marking segment junctions.

FIGURE 40. Using sinusoidal segments with constant speed intermediate interval

$$\begin{aligned}
&= \left(k_1 \cdot \frac{2}{\pi} \cdot \sin\left(\left(\frac{t}{k_1} \cdot \frac{\pi}{2} - \frac{\pi}{2}\right) + 1\right) \right) / f && t \leq k_1 \\
east(t) &= \left(\frac{k_1}{\pi/2} + t - k_1 \right) / f && k_1 \leq t \leq k_2 \\
&= \left(\frac{k_1}{\pi/2} + k_2 - k_1 + \left((1 - k_2) \cdot \frac{2}{\pi} \right) \sin\left(\frac{t - k_2}{1.0 - k_2} \cdot \frac{\pi}{2}\right) \right) / f && k_2 \leq t
\end{aligned} \tag{EQ 45}$$

where $f = k_1 \cdot 2/\pi + k_2 - k_1 + (1.0 - k_2) \cdot 2/\pi$

```

float ease(float t, float k1, float k2)
{
    float t1,t2;
    float f,s;

    f = k1 *2/3.1415926535 + k2 - k1 + (1.0-k2)*2/PI;

    if (t < k1) {
        s = k1*(2/PI)*(sin((t/k1)*PI/2-PI/2)+1);
    }
    else if (t < k2) {
        s = (2*k1/PI + t-k1);
    }
    else {
        s = 2*k1/PI + k2-k1 + ((1-k2)*(2/PI))*sin(((t-k2)/(1.0-k2))*PI/2);
    }
    return (s/f);
}

```

FIGURE 41. Code which implements ease-in/ease-out by using sinusoidal ease-in, followed by constant velocity and sinusoidal ease-out.

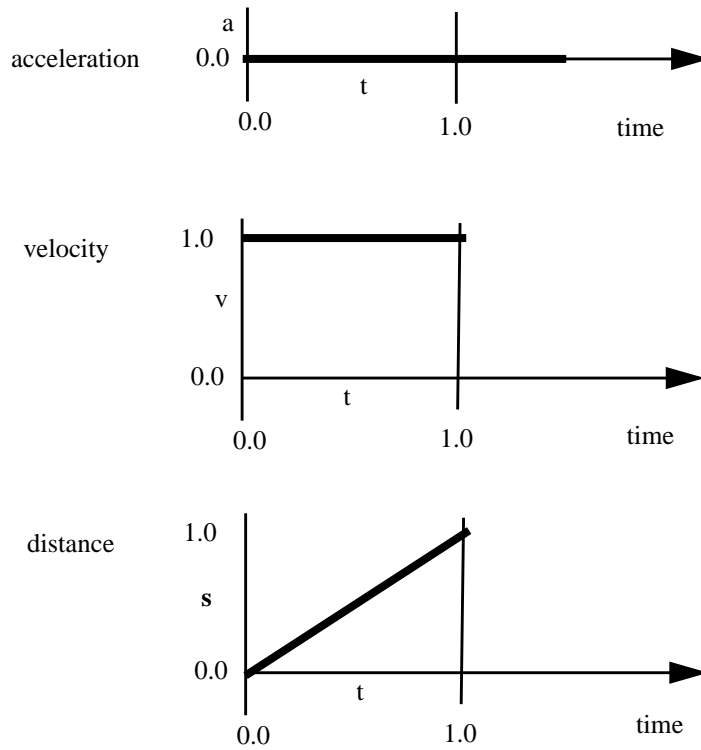
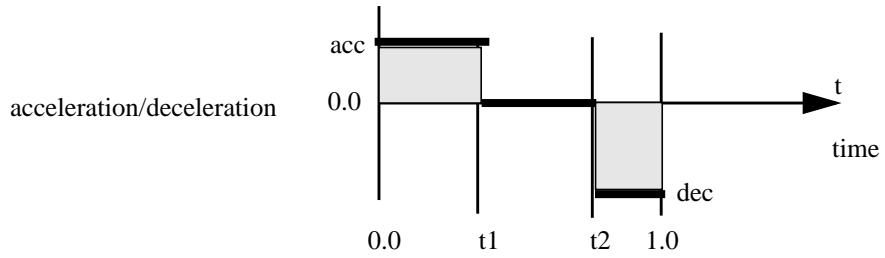


FIGURE 42. Acceleration, velocity, and distance curves for constant speed.



$$\begin{aligned}
 a &= \text{acc} & 0 < t < t1 \\
 a &= 0.0 & t1 < t < t2 \\
 a &= \text{dec} & t2 < t < 1.0
 \end{aligned}$$

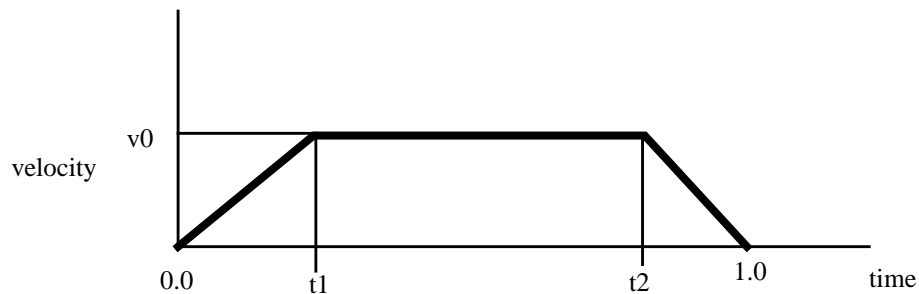
FIGURE 43. Acceleration/deceleration graph.

1.0 = area under leading linear ramp +
 area under middle constant velocity interval +
 area under ending linear ramp

(EQ 46)

$$1.0 = \frac{1}{2} \cdot v0 \cdot t1 + v0 \cdot (t2 - t1) + \frac{1}{2} \cdot v0 \cdot (1.0 - t2)$$

$$v_0 = \frac{2.0}{(t_2 - t_1 + 1.0)} \quad . \quad (\text{EQ 47})$$

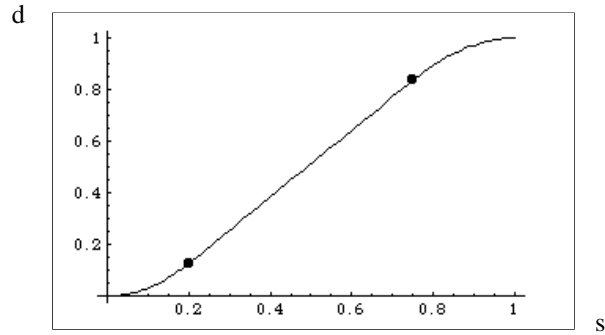


$$v = v_0 \cdot \frac{t}{t_1} \quad 0.0 < t < t_1$$

$$v = v_0 \quad t_1 < t < t_2$$

$$v = v_0 \cdot \left(1.0 - \frac{t - t_2}{1.0 - t_2}\right) \quad t_2 < t < 1.0$$

FIGURE 44. Velocity-time curve.

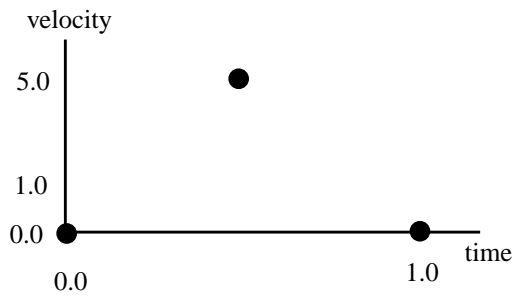


$$d = v_0 \cdot \frac{t^2}{2 \cdot t_1} \quad 0.0 < t < t_1$$

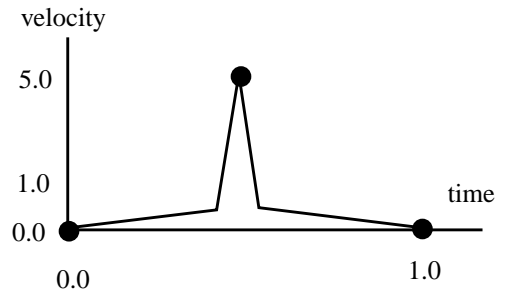
$$d = v_0 \cdot \frac{t_1}{2} + v_0 \cdot (t - t_1) \quad t_1 < t < t_2$$

$$d = v_0 \cdot \frac{t_1}{2} + v_0 \cdot (t_2 - t_1) + \left(v_0 - \frac{\left(v_0 \cdot \frac{t - t_2}{1 - t_2} \right)}{2} \right) \cdot (t - t_2) \quad t_2 < t < 1.0$$

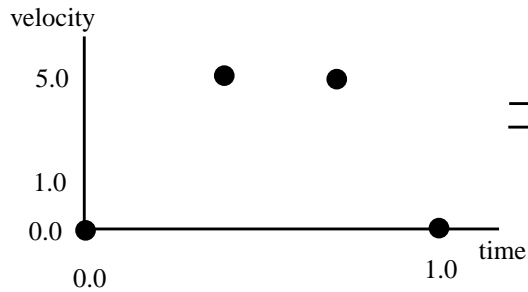
FIGURE 45. Distance-time function.



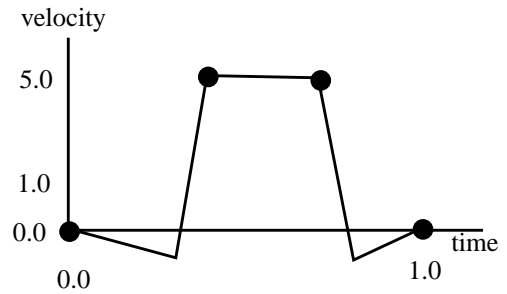
User specified velocities



Possible solution to enforce total distance covered equal to one

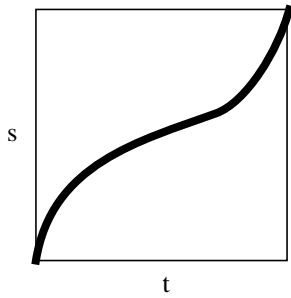


User specified velocities

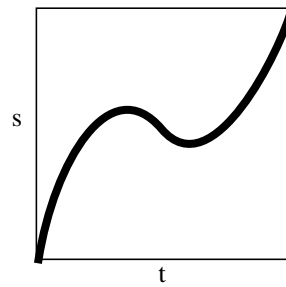


Possible solution to enforce total distance covered (signed area under the curve) equal to one

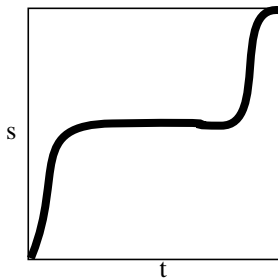
FIGURE 46. Some non-intuitive results of user-specified values on the velocity-time curve.



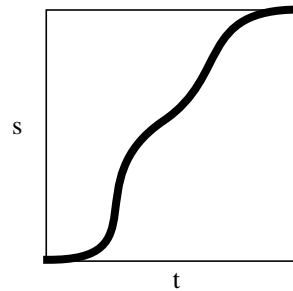
a) starts and ends abruptly



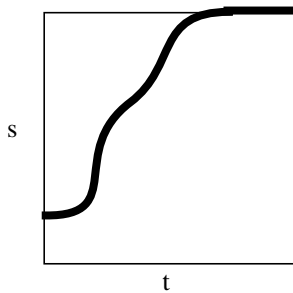
b) backs up



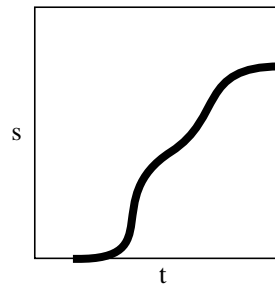
c) stalls



d) smoothly starts and stops



e) starts part way along the curve and gets to the end before $t=1.0$



f) waits awhile before starting and doesn't get all the way to the end

FIGURE 47. Sample distance-time functions.

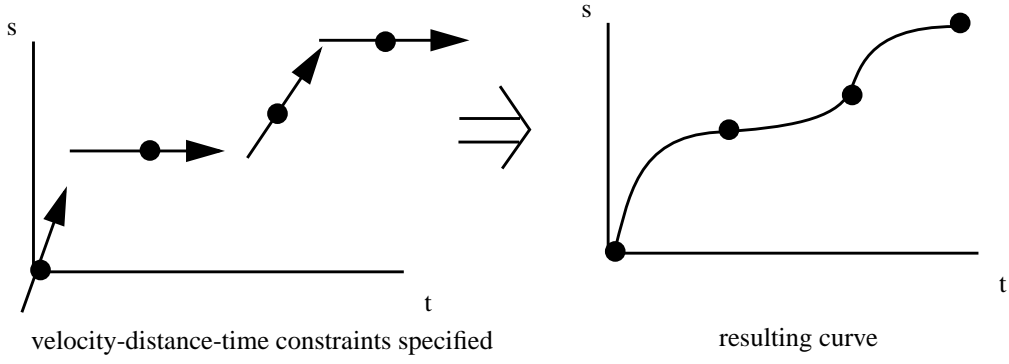
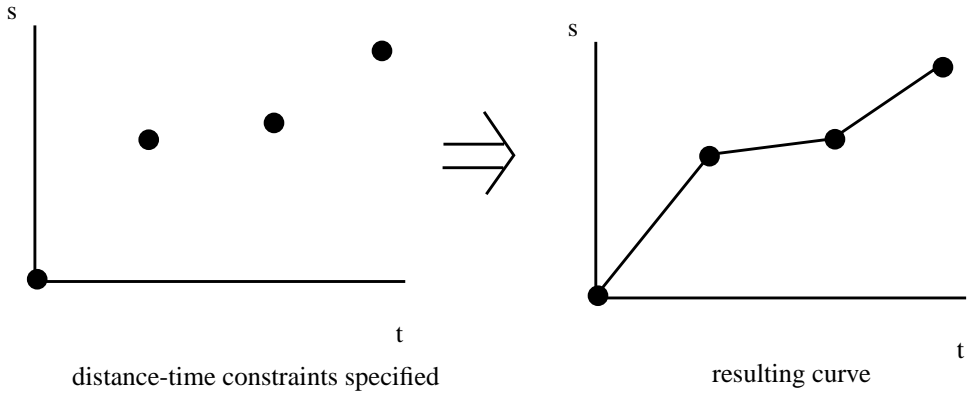


FIGURE 48. Specifying motion constraints.

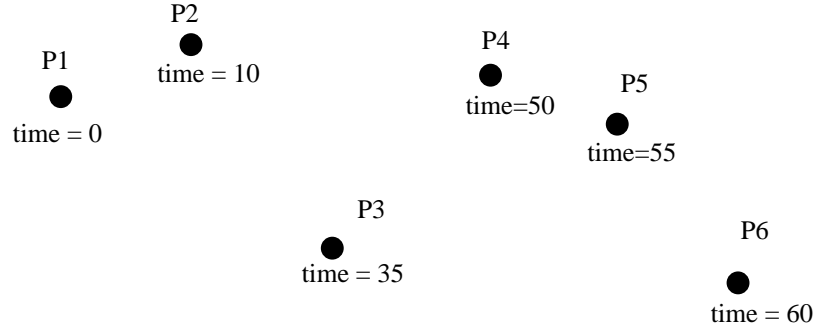


FIGURE 49. Position-time constraints.

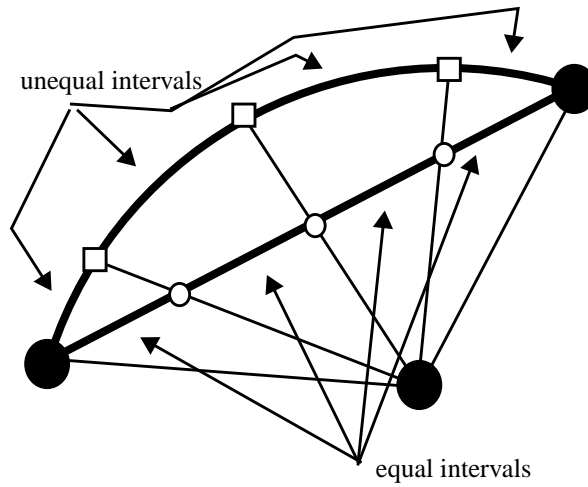
$$P(t) = \sum_{i=1}^{n+1} B_i \cdot N_{i,k}(t) \quad (\text{EQ 48})$$

$$\begin{aligned}
 P_1 &= N_{1,k}(t_1) \cdot B_1 + N_{2,k}(t_1) \cdot B_2 + \dots + N_{n+1,k}(t_1) \cdot B_{n+1} \\
 P_2 &= N_{1,k}(t_2) \cdot B_1 + N_{2,k}(t_2) \cdot B_2 + \dots + N_{n+1,k}(t_2) \cdot B_{n+1} \\
 &\dots \\
 P_j &= N_{1,k}(t_j) \cdot B_1 + N_{2,k}(t_j) \cdot B_2 + \dots + N_{n+1,k}(t_j) \cdot B_{n+1}
 \end{aligned} \quad (\text{EQ 49})$$

$$P = N \cdot B \quad (\text{EQ 50})$$

$$B = N^{-1} \cdot P \quad (\text{EQ 51})$$

$$\begin{aligned}
 P &= N \cdot B \\
 N^T \cdot P &= N^T \cdot N \cdot B \\
 [N^T \cdot N]^{-1} \cdot N^T \cdot P &= B
 \end{aligned} \quad (\text{EQ 52})$$



- linearly interpolated intermediate points
- projection of intermediate points onto circle

FIGURE 50. Equally spaced linear interpolation of straight-line path between two points on a circle generate unequal spacing of points after projecting onto a circle.

$$\cos \theta = q1 \cdot q2 = s1 \cdot s2 + v1 \cdot v2$$

(EQ 53)

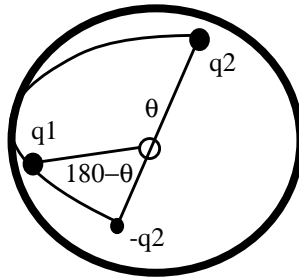


FIGURE 51. The closer of the two representations of orientation is the better choice to use in interpolation.

$$Slerp(q1, q2, u) = ((\sin((1 - u) \cdot \theta))/(\sin\theta)) \cdot q1 + (\sin(u \cdot \theta))/(\sin\theta) \cdot q2 \quad \text{(EQ 54)}$$

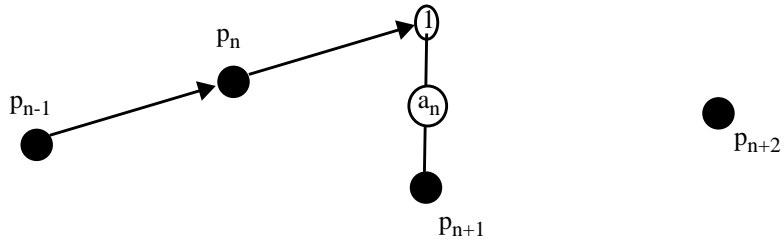


FIGURE 52. Finding the control point after p_n .

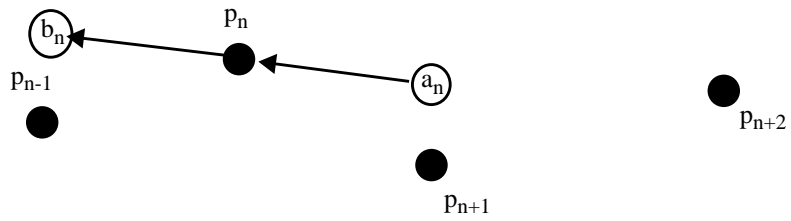


FIGURE 53. Finding the control point before p_n .

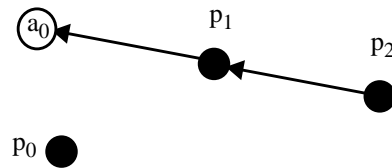


FIGURE 54. Constructing the first interior control point.

$$b_n' = p_n + k \cdot (b_n - p_n)$$

(EQ 55)

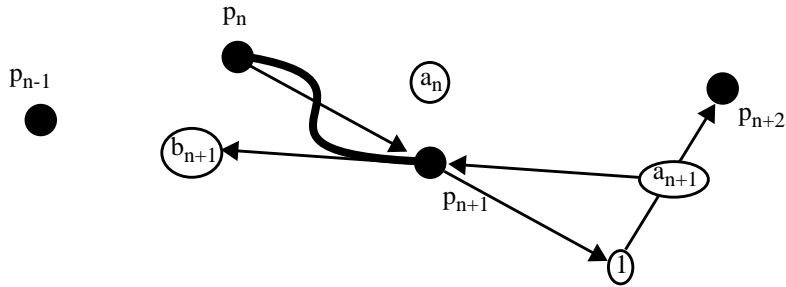


FIGURE 55. Construction of b_{n+1} and the resulting curve.

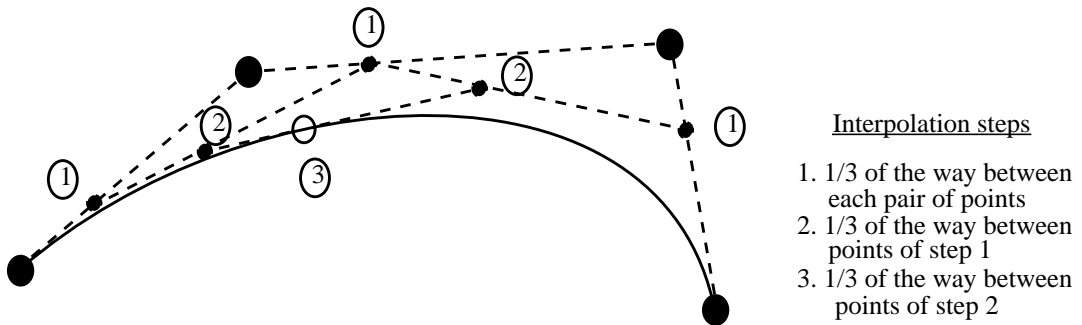


FIGURE 56. De Casteljau construction of point on cubic Bezier segment at 1/3 (the point labeled '3').

$p1 = \text{slerp}(q_n, a_n, 1/3)$
 $p2 = \text{slerp}(a_n, b_{n+1}, 1/3)$
 $p3 = \text{slerp}(b_{n+1}, q_{n+1}, 1/3)$
 $p12 = \text{slerp}(p1, p2, 1/3)$
 $p23 = \text{slerp}(p2, p3, 1/3)$
 $p = \text{slerp}(p12, p23, 1/3)$

$\text{temp} = \text{slerp}(q_n, a_n, 1/2) = q_n + a_n$
 $p1 = \text{slerp}(q_n, \text{temp}, 1/2) = q_n + \text{temp}$
 $\text{temp} = \text{slerp}(a_n, b_{n+1}, 1/2) = a_n + b_{n+1}$
 $p2 = \text{slerp}(a_n, \text{temp}, 1/2) = a_n + \text{temp}$
 $\text{temp} = \text{slerp}(b_{n+1}, q_{n+1}, 1/2) = b_{n+1} + q_{n+1}$
 $p3 = \text{slerp}(b_{n+1}, \text{temp}, 1/2) = b_{n+1} + \text{temp}$
 $\text{temp} = \text{slerp}(p1, p2, 1/2) = p1 + p2$
 $p12 = \text{slerp}(p1, \text{temp}, 1/2) = p1 + \text{temp}$
 $\text{temp} = \text{slerp}(p2, p3, 1/2) = p2 + p3$
 $p23 = \text{slerp}(p2, \text{temp}, 1/2) = p2 + \text{temp}$
 $\text{temp} = \text{slerp}(p12, p23, 1/2) = p12 + p23$
 $p = \text{slerp}(p12, \text{temp}, 1/2) = p12 + \text{temp}$

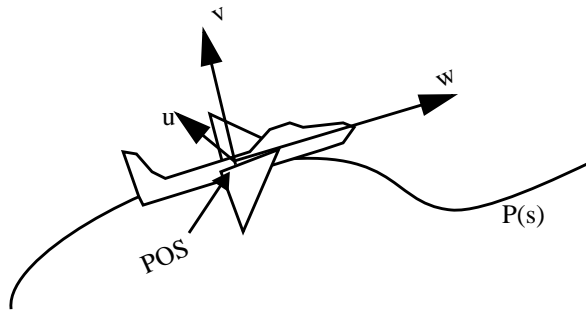


FIGURE 57. Camera-based local coordinate system.

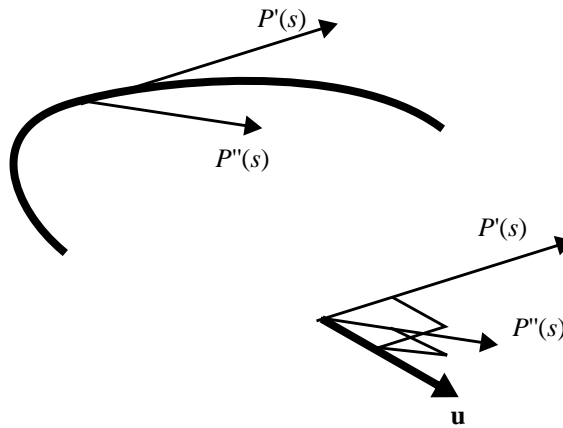


FIGURE 58. The derivatives at a point along the curve.

$$\begin{aligned}
 w &= P'(s) \\
 u &= P'(s) \times P''(s) \\
 v &= w \times u
 \end{aligned}
 \tag{EQ 56}$$

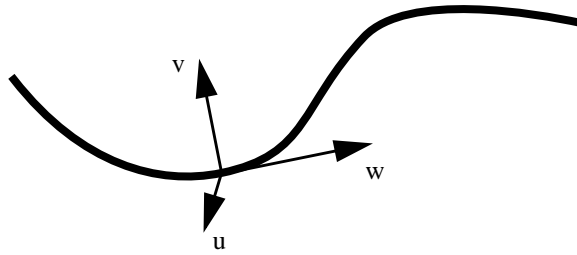
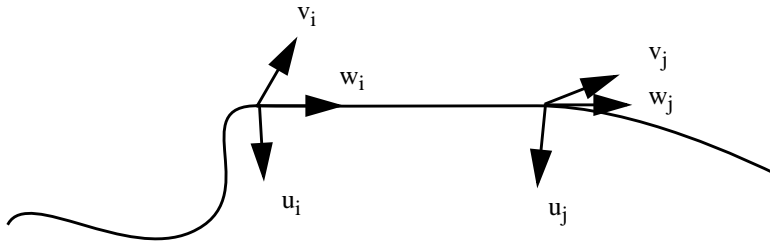
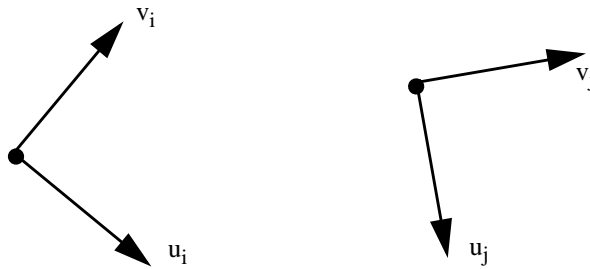


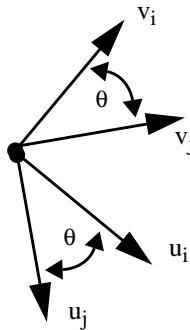
FIGURE 59. Frenet Frame at a point along a curve.



a) Frenet frames on boundary of undefined Frenet frame segment



b) The two frames sighted down the (common) w vector



c) The two frames superimposed to identify angular difference

FIGURE 60. Interpolating Frenet frames to determine the undefined segment.

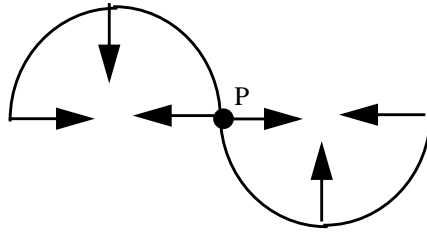


FIGURE 61. The curvature vector, defined everywhere but discontinuous, instantaneously switches direction at point P.

$$\begin{aligned}
 w &= POS - COI \\
 u &= w \times \text{y-axis} \\
 v &= u \times w
 \end{aligned}
 \tag{EQ 57}$$

:

$$\begin{aligned}
 w &= C(s) - P(s) \\
 u &= (U(s) - P(s)) \times w \\
 v &= w \times u
 \end{aligned}
 \tag{EQ 58}$$

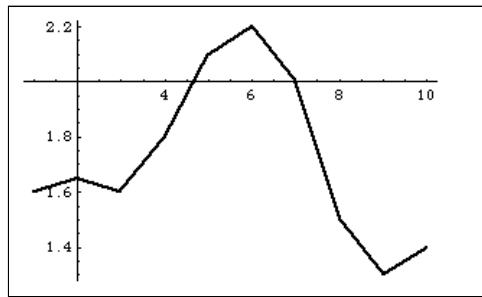


FIGURE 62. Sample data for path smoothing.

$$P'_i = \frac{P_i + \frac{P_{i-1} + P_{i+1}}{2}}{2} = \frac{1}{4} \cdot P_{i-1} + \frac{1}{2} \cdot P_i + \frac{1}{4} \cdot P_{i+1} \quad (\text{EQ 59})$$

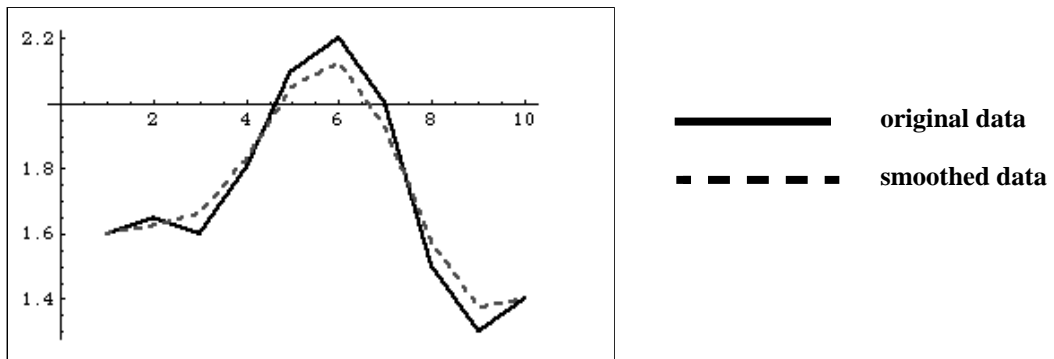


FIGURE 63. Sample data smoothed by linear interpolation.

$$P(u) = a \cdot u^3 + b \cdot u^2 + c \cdot u + d \quad (\text{EQ 60})$$

$$\begin{aligned}
 P_{i-2} &= P(0) = d \\
 P_{i-1} &= P(1/4) = a \cdot \frac{1}{64} + b \cdot \frac{1}{16} + c \cdot \frac{1}{4} + d \\
 P_{i+1} &= P(3/4) = a \cdot \frac{27}{64} + b \cdot \frac{9}{16} + c \cdot \frac{3}{4} + d \\
 P_{i+2} &= a + b + c + d
 \end{aligned}
 \tag{EQ 61}$$

$$\tag{EQ 62}$$

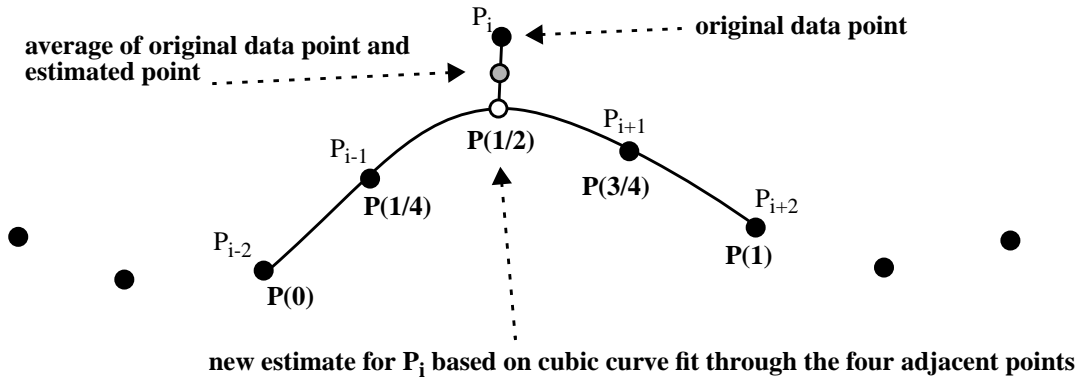


FIGURE 64. Smoothing data by cubic interpolation.

new estimate for P_i based on cubic curve fit through the four adjacent points

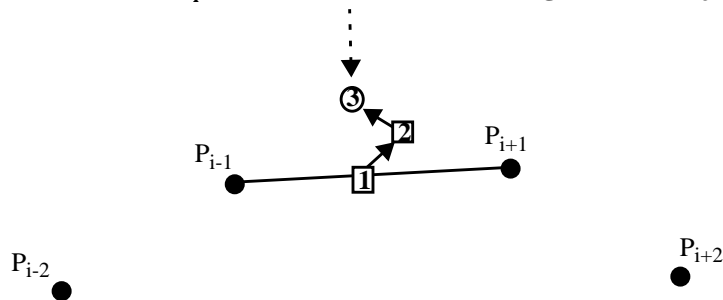


FIGURE 65. Geometric construction of a cubic estimate for smoothing a data point.

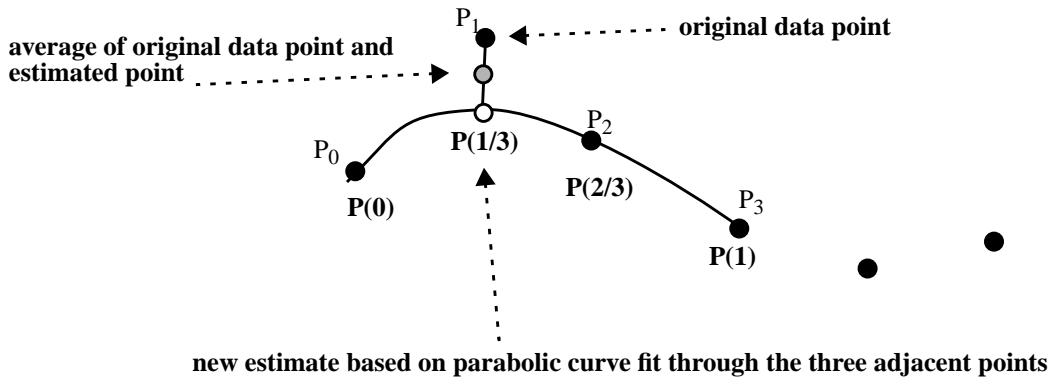
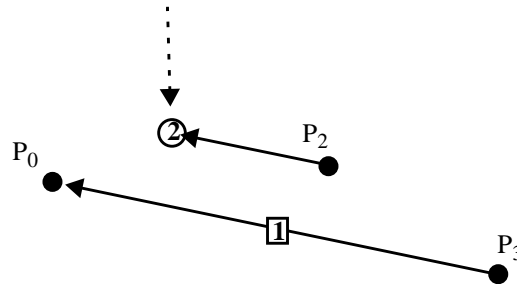


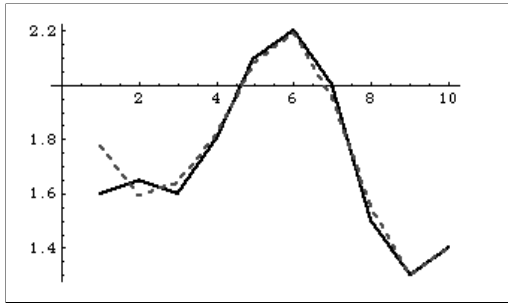
FIGURE 66. Smoothing data by parabolic interpolation.

new estimate for P_1 based on parabolic curve fit through the three adjacent points

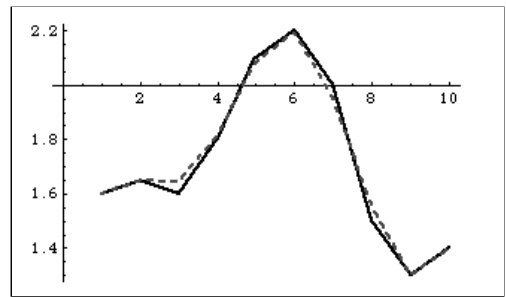


- 1) construct vector from P_3 to P_0
- 2) add $1/3$ of the vector to P_2
- 3) (not shown) average estimated point with original data point

FIGURE 67. Geometric construction of a parabolic estimate for smoothing a data point.

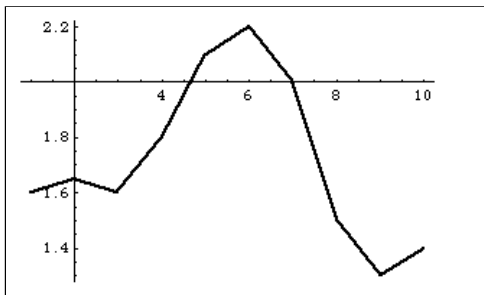


a) cubic smoothing with parabolic end conditions

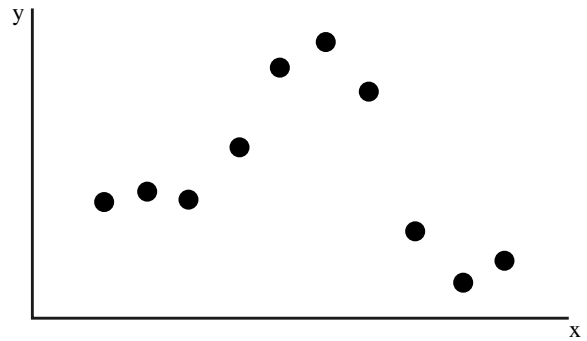


b) cubic smoothing without smoothing the end points

FIGURE 68. Sample data smoothed with cubic interpolation.



a) original curve



b) data points of original curve

FIGURE 69. Sample function to be smoothed.

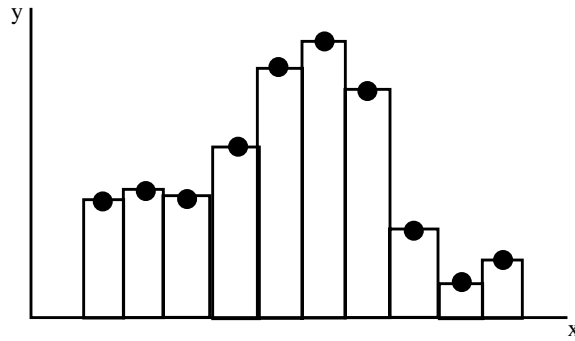
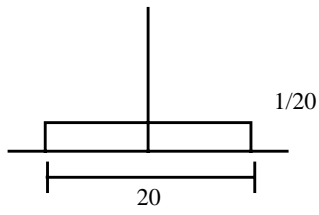
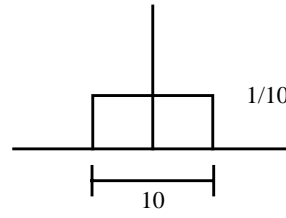


FIGURE 70. Step function defined by data points of original curve.

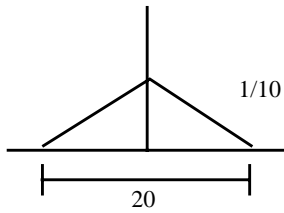
$$P(x) = \int_{-s}^s f(x+u) \cdot g(u) du \quad . \quad (\text{EQ 63})$$



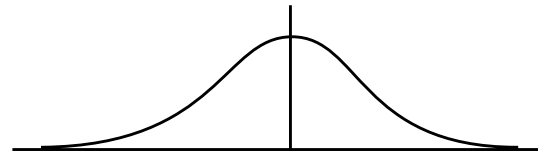
a) wide box



b) box

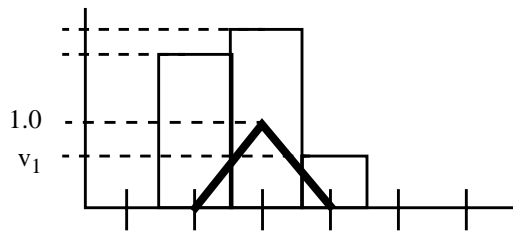


c) tent

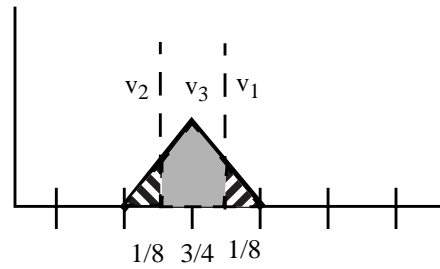


d) Gaussian $\frac{1}{a \cdot \sqrt{2 \cdot \pi}} \cdot e^{-\frac{(x-b)^2}{2 \cdot a^2}}$

FIGURE 71. Sample smoothing kernels.



a) smoothing kernel superimposed over step function



b) areas of tent kernel under the different step function values

$$V = \frac{1}{8} \cdot v_1 + \frac{3}{4} \cdot v_2 + \frac{1}{8} \cdot v_3$$

c) computation of smoothed value by applying area weights to step function values

FIGURE 72. Example of a tent-shaped smoothing filter.

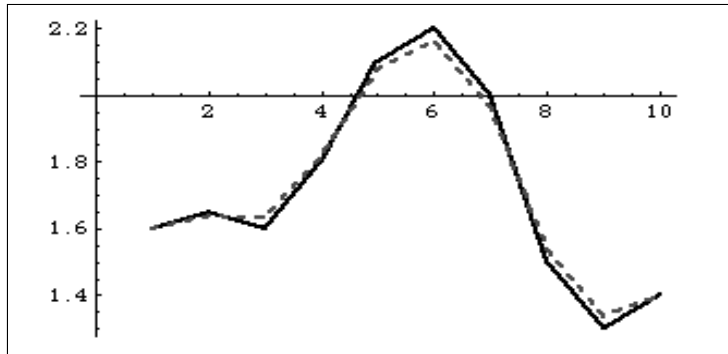


FIGURE 73. Sample data smoothed with convolution using a tent kernel.

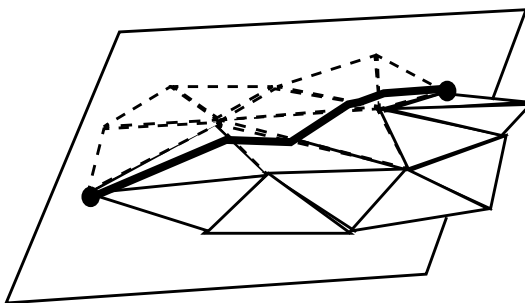


FIGURE 74. Determining a path along a polygonal surface mesh by using plane intersection.

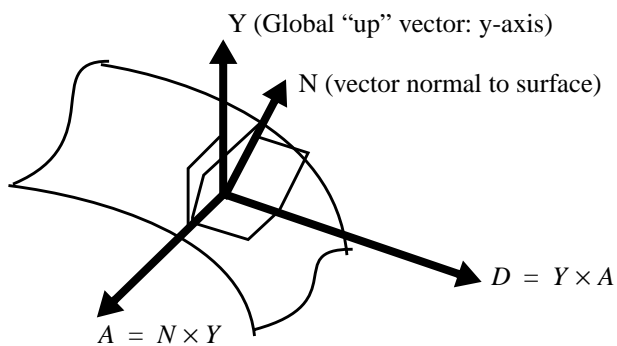


FIGURE 75. Calculating the downhill vector, D.

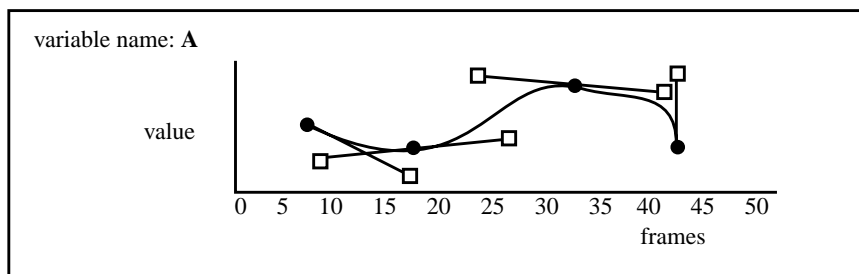
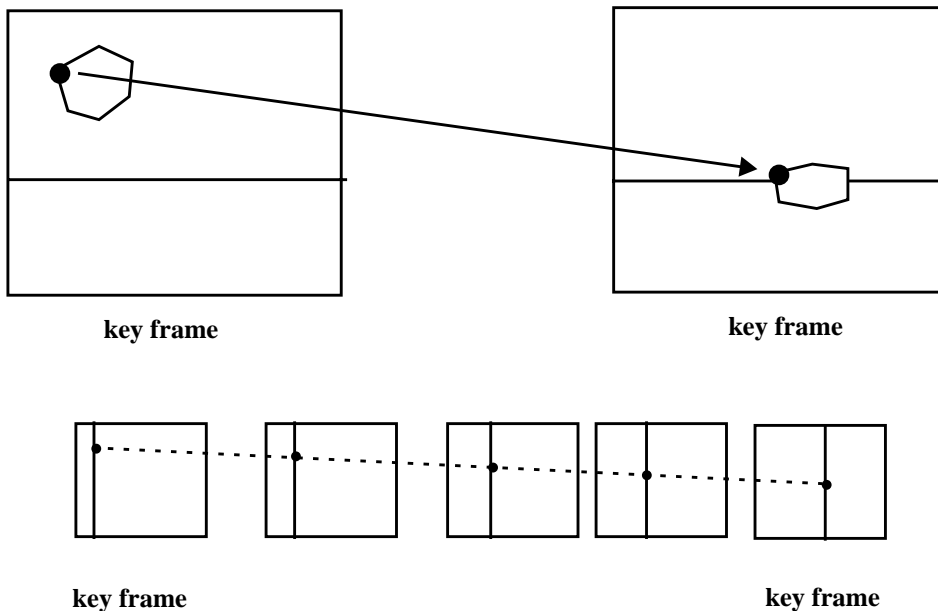


FIGURE 76. Simple interface for specifying interpolation of key values.



keys and three intermediate frames with linear interpolation of a single point (with reference lines showing the progression of the interpolation in x and y)

FIGURE 77. Simple key frames in which each curve of a frame has the same number of points as its counterpart in the other frame. Frames showing linear interpolation of a single point (with reference lines) in intermediate frames.

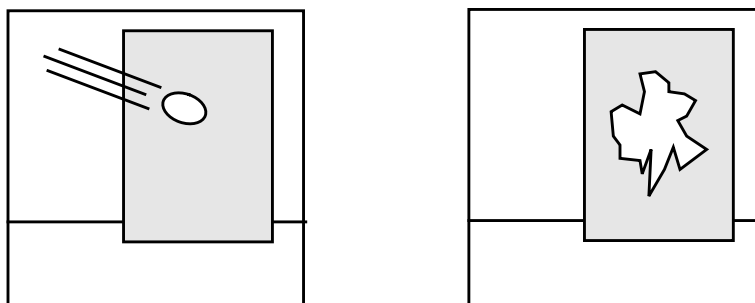
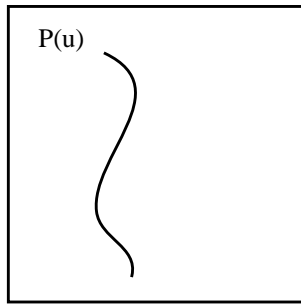
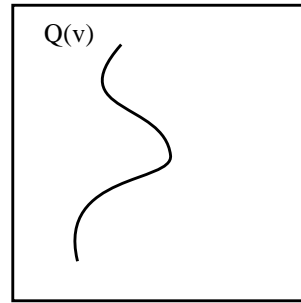


FIGURE 78. Egg splatting against a wall whose shape must be interpolated.



a) frame f1



b) frame f2

FIGURE 79. Two frames showing a curve to be interpolated.

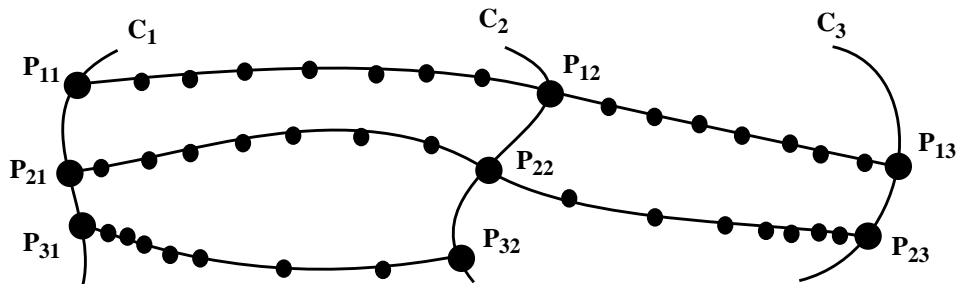


FIGURE 80. Moving point constraints.

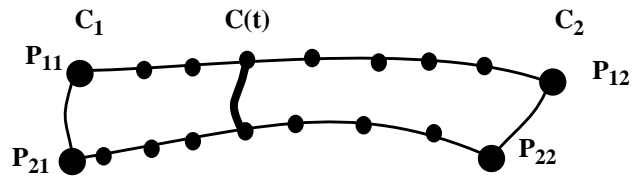
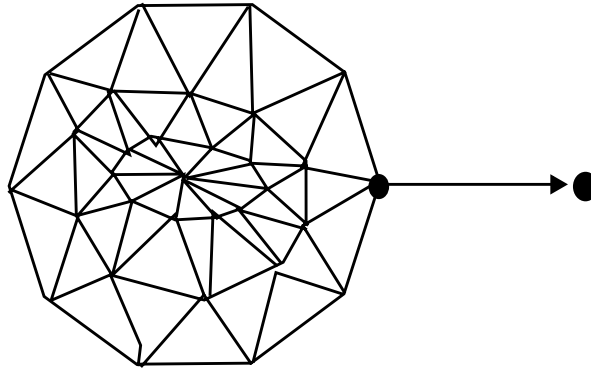
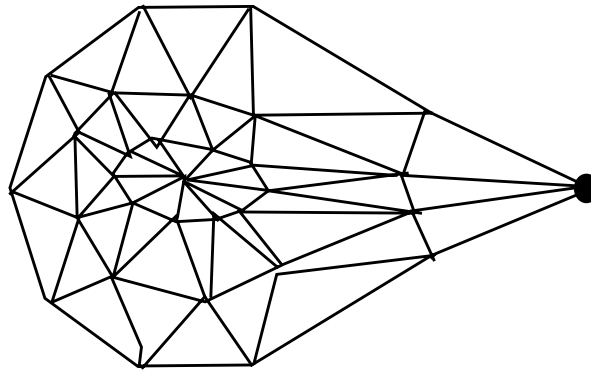


FIGURE 81. Patch defined by interpolation constraints.



a) Displacement of seed vertex



b) Attenuated displacement propagated to adjacent vertices.

FIGURE 82. Warping of object vertices.

$$\begin{aligned}
 S(i) &= 1.0 - \left(\frac{i}{n+1}\right)^{k+1} & k \geq 0 \\
 &= \left(1.0 - \left(\frac{i}{n+1}\right)\right)^{-k} & k < 0
 \end{aligned}
 \tag{EQ 64}$$

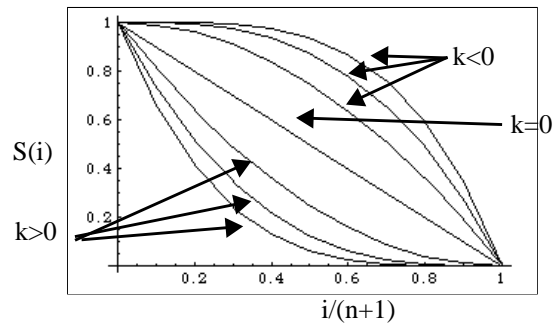


FIGURE 83. Power functions.

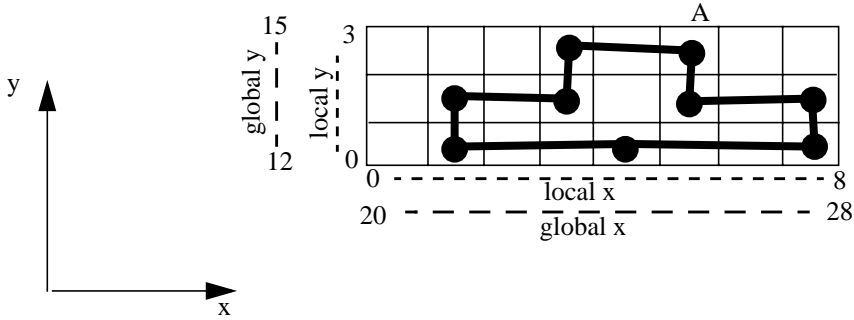


FIGURE 84. Initial 2D coordinate grid.

$$\begin{aligned}
 P_{u0} &= (1-u) \cdot P_{00} + u \cdot P_{10} \\
 P_{u1} &= (1-u) \cdot P_{01} + u \cdot P_{11} \\
 P_{uv} &= (1-v) \cdot P_{u0} + v \cdot P_{u1} \\
 &= ((1-u) \cdot (1-v) \cdot P_{00} + (1-u) \cdot v \cdot P_{01} + u \cdot (1-v) \cdot P_{10} + u \cdot v \cdot P_{11})
 \end{aligned}$$

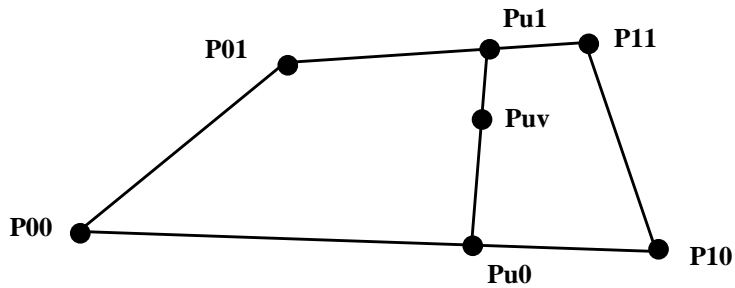


FIGURE 85. Bilinear interpolation.

$$P = 0.6 \cdot 0.7 \cdot P00 + 0.6 \cdot 0.3 \cdot P01 + 0.4 \cdot 0.7 \cdot P10 + 0.4 \cdot 0.3 \cdot P11$$

(EQ 65)

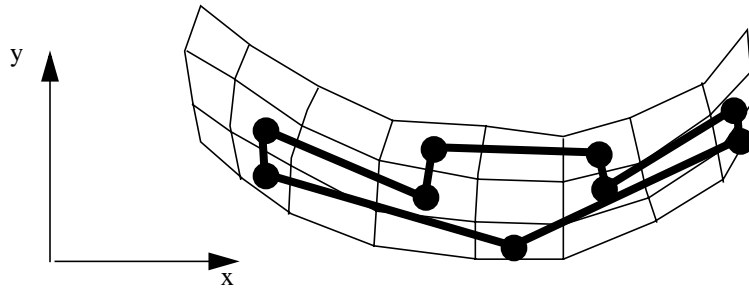


FIGURE 86. 2D Grid Deformation.

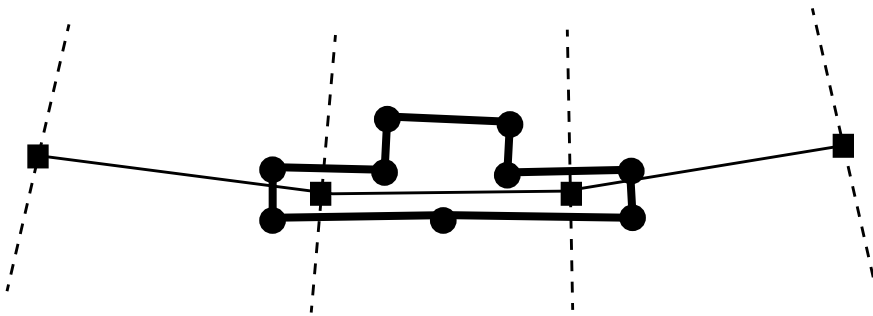


FIGURE 87. Polyline drawn through object. Bisectors and perpendiculars are drawn as dashed lines.

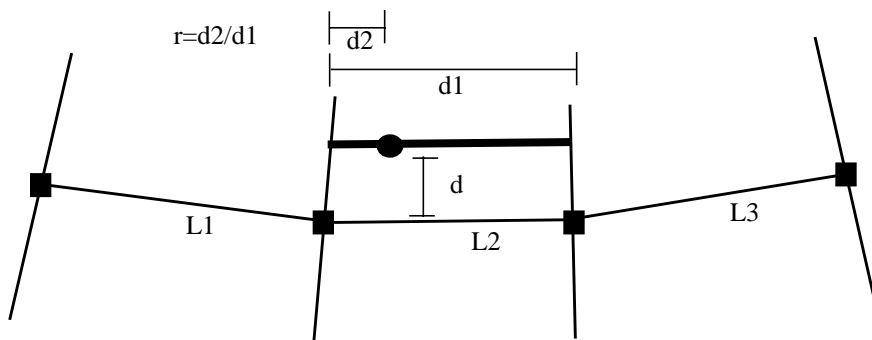


FIGURE 88. Measurements used to map an object vertex to a polyline.

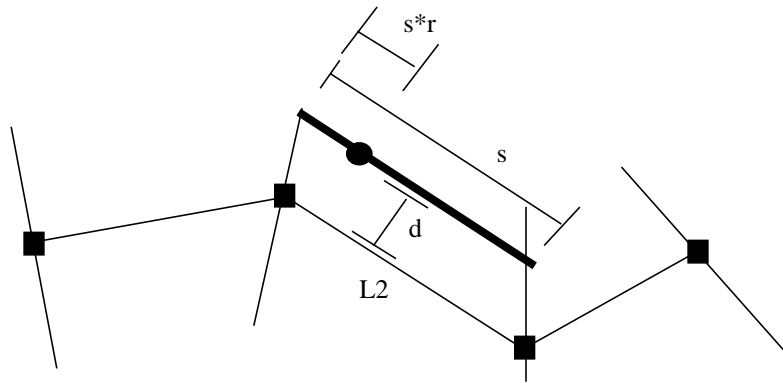
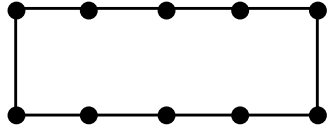
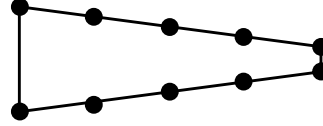


FIGURE 89. Remapping of an object vertex relative to a deformed polyline; refer to Figure 89.



a) original object



b) tapered object

$$\begin{aligned}
 x' &= x \\
 y' &= f(x)
 \end{aligned}
 \quad
 \begin{aligned}
 \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & f(x) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \\
 P' &= M(P) \cdot P
 \end{aligned}$$

FIGURE 90. Global tapering.

$$\begin{aligned}
 x' &= x \cdot \cos(f(x)) - y \cdot \sin(f(x)) \\
 y' &= x \cdot \sin(f(x)) + y \cdot \cos(f(x)) \\
 z' &= z
 \end{aligned}$$

FIGURE 91. Twist about an axis.

y_0 - center of bend $1/k$ - radius of bend $y_{min}:y_{max}$ - bend region

$$\hat{y} = \begin{cases} y_{min} & y \leq y_{min} \\ y & y_{min} < y < y_{max} \\ y_{max} & y \geq y_{max} \end{cases}$$

$$\begin{aligned} \theta &= k \cdot (\hat{y} - y_0) \\ C_\theta &= \cos \theta \\ S_\theta &= \sin \theta \end{aligned}$$

$$x' = x$$

$$y' = \begin{cases} -S_\theta \cdot z - \frac{1}{k} + y_0 & y_{min} \leq y \leq y_{max} \\ -\left(S_\theta \cdot \left(z - \frac{1}{k}\right)\right) + y_0 + C_\theta \cdot (y - y_{min}) & y < y_{min} \\ \left(-\left(S_\theta \cdot \left(z - \frac{1}{k}\right)\right) + y_0 + C_\theta \cdot (y - y_{max})\right) & y > y_{max} \end{cases} \quad \text{(EQ 66)}$$

$$z' = \begin{cases} -C_\theta \cdot z - \frac{1}{k} + \frac{1}{k} & y_{min} \leq y \leq y_{max} \\ -\left(C_\theta \cdot \left(z - \frac{1}{k}\right)\right) + \frac{1}{k} + S_\theta \cdot (y - y_{min}) & y < y_{min} \\ \left(-\left(C_\theta \cdot \left(z - \frac{1}{k}\right)\right) + \frac{1}{k} + S_\theta \cdot (y - y_{max})\right) & y > y_{max} \end{cases}$$

FIGURE 92. Examples of global deformations

$$s = (T \times U) \bullet (P - P0) / ((T \times U) \bullet S) \quad \text{(EQ 67)}$$

$$t = (U \times S) \bullet (P - P0) / ((U \times S) \bullet T) \quad \text{(EQ 68)}$$

$$u = (S \times T) \bullet (P - P0) / ((S \times T) \bullet U) \quad \text{(EQ 69)}$$

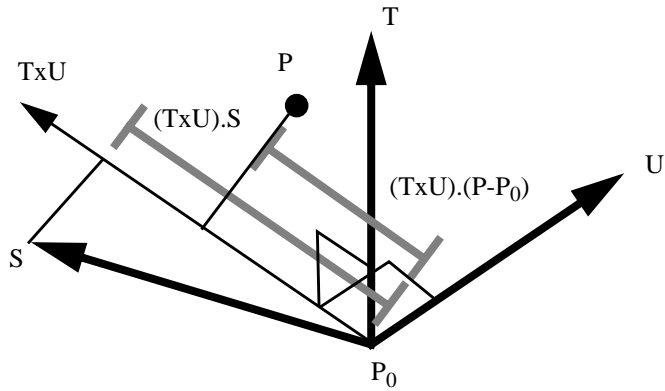


FIGURE 93. Initial local coordinate system for FFDs.

$$P = P_0 + s \cdot S + t \cdot T + u \cdot U \quad (\text{EQ 70})$$

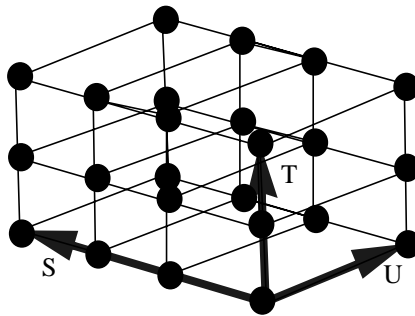


FIGURE 94. Grid of control points.

$$P_{ijk} = P_0 + \frac{i}{l} \cdot S + \frac{j}{m} \cdot T + \frac{k}{n} \cdot U \quad (\text{EQ 71})$$

$$P(s, t, u) = \sum_{i=0}^l \binom{l}{i} (1-s)^{l-i} s^i \cdot \left(\sum_{j=0}^m \binom{m}{j} (1-t)^{m-j} t^j \cdot \left(\sum_{k=0}^n \binom{n}{k} (1-u)^{n-k} u^k P_{ijk} \right) \right) \quad (\text{EQ 72})$$

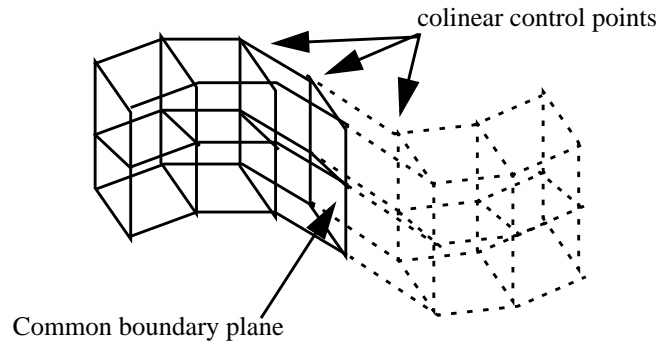


FIGURE 95. C^1 continuity between adjacent control grids.

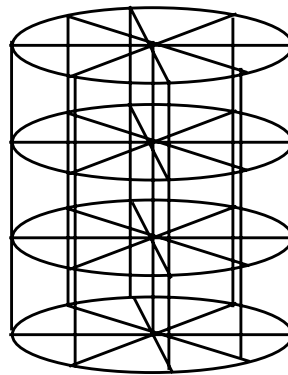
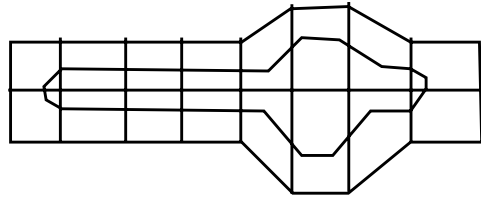
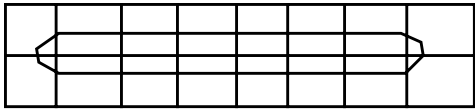
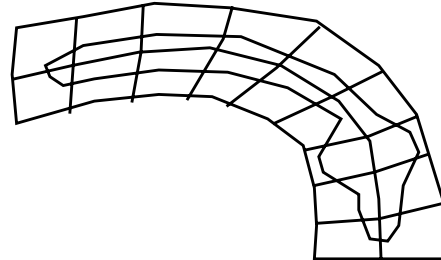
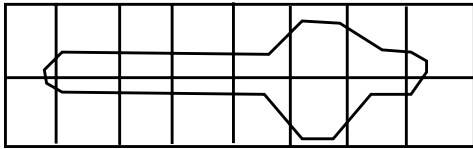


FIGURE 96. Cylindrical grid.



a) Bulging



b) Bending

FIGURE 97. Sequential FFDs.

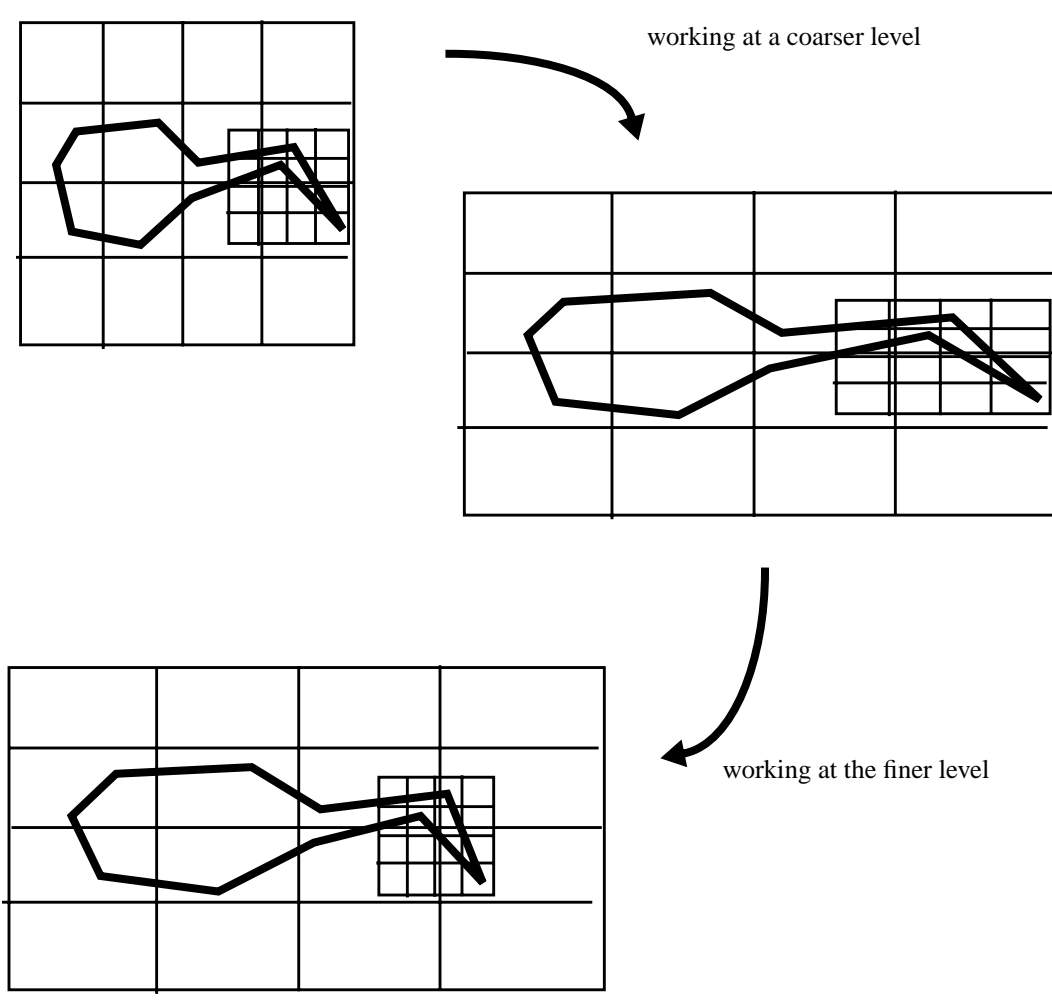
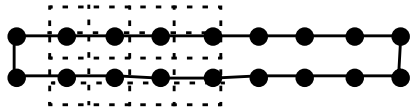
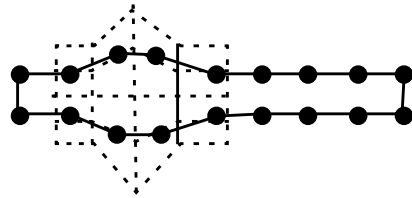


FIGURE 98. Simple example of hierarchical FFDs.



a) undeformed object



b) deformed object

FIGURE 99. Deformation tool applied to an object.

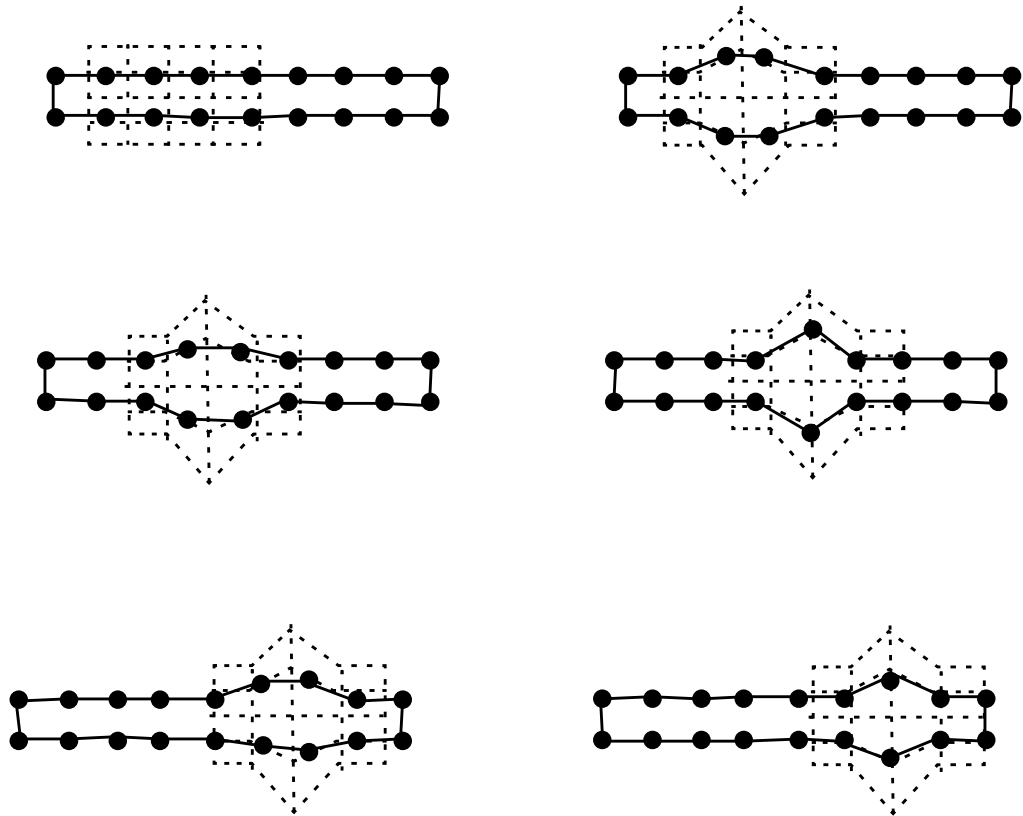
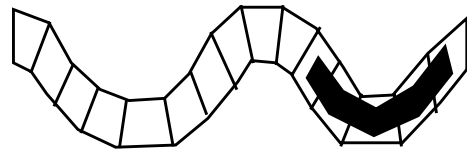
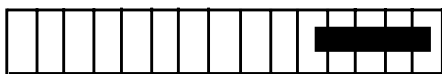
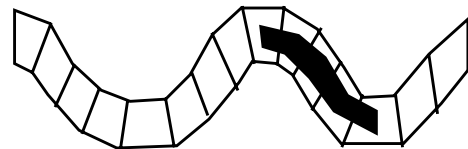
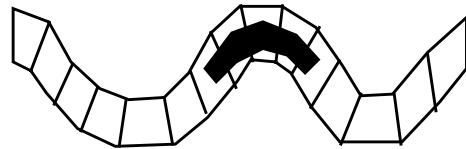
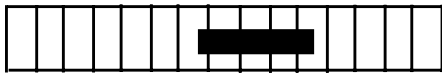
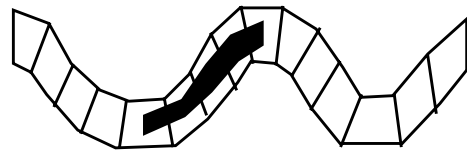
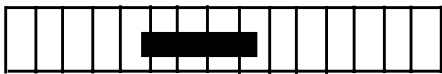
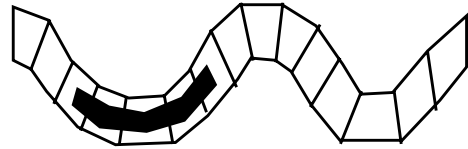
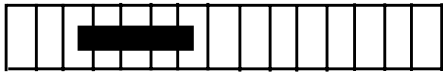
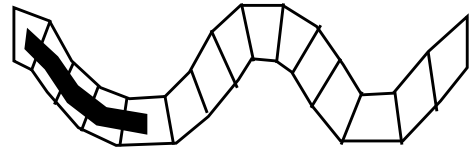
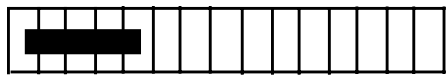


FIGURE 100. Deformation by translating the deformation tool relative to an object.



a) Object traversing the logical FFD coordinate space

b) Object traversing the distorted space

FIGURE 101. Deformation of an object by passing through FFD space.

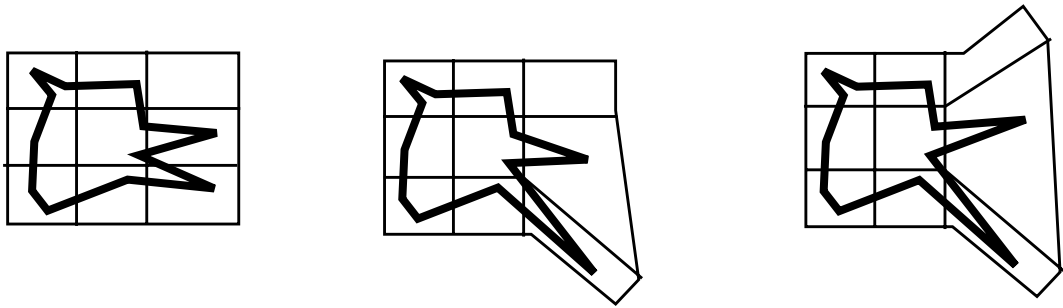
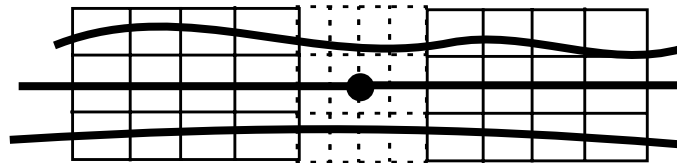
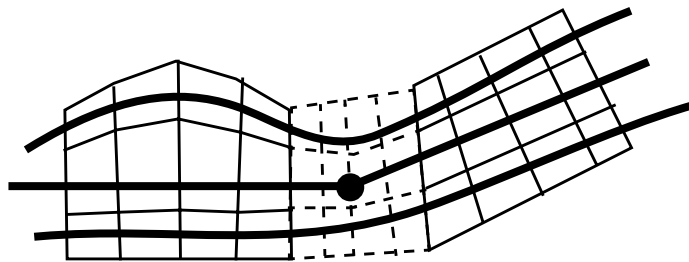


FIGURE 102. Using FFD type of mesh to animate a figure's head.

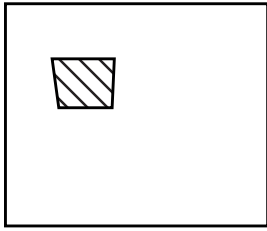


a) initial configuration

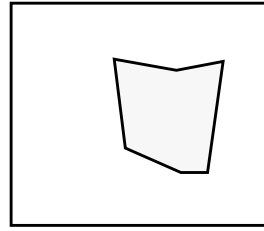


b) Surface distorted after joint articulation

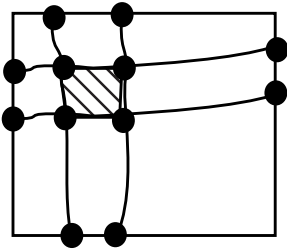
FIGURE 103. Using FFD to deform a surface around an articulated joint.



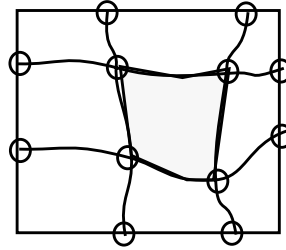
a) Image A



b) Image B



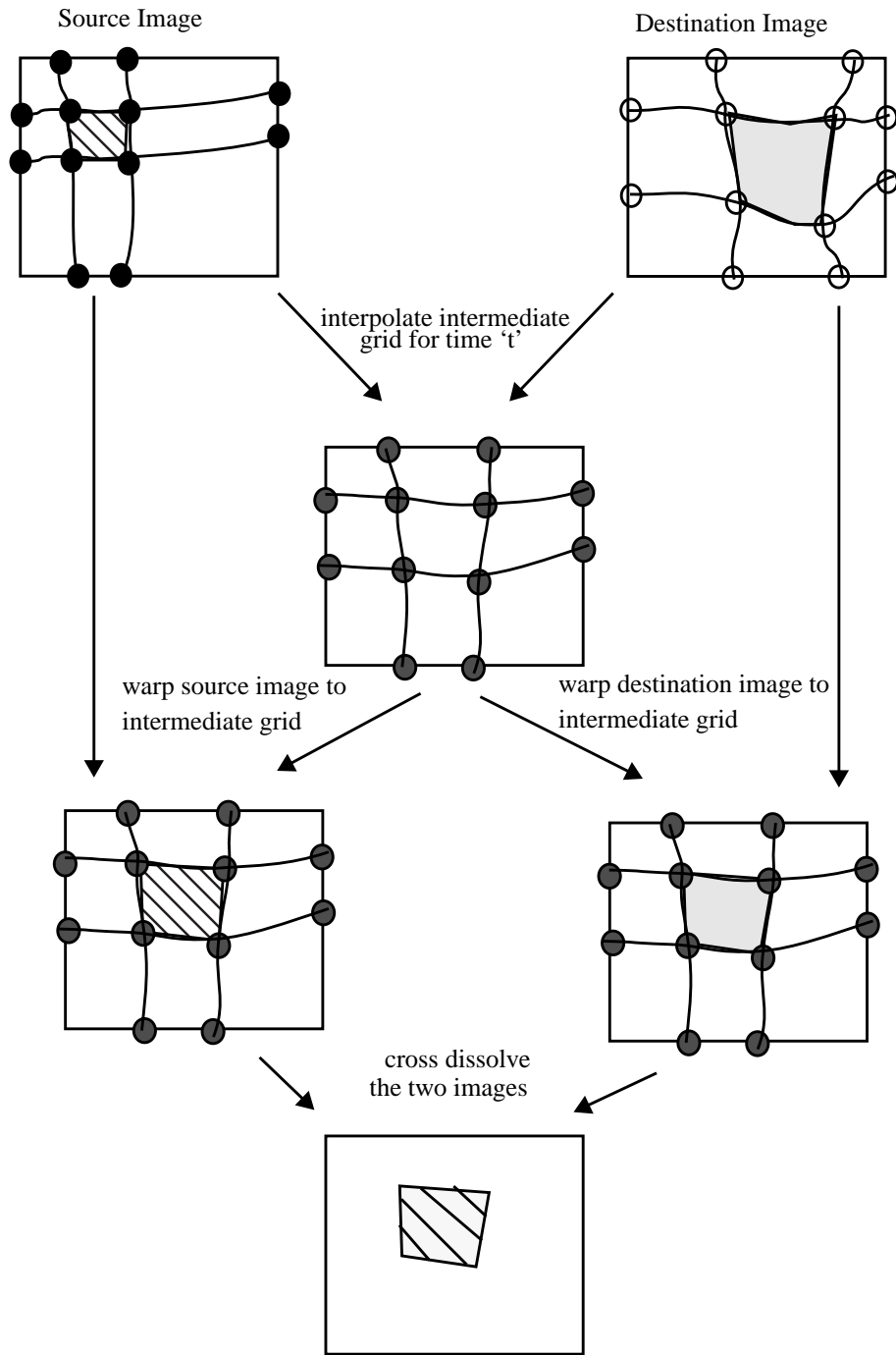
c) Image A with grid points and curves defined



d) Image B with grid points and curves defined

FIGURE 104. Sample grid definitions.

FIGURE 105. Interpolating to intermediate grid and cross-dissolve.



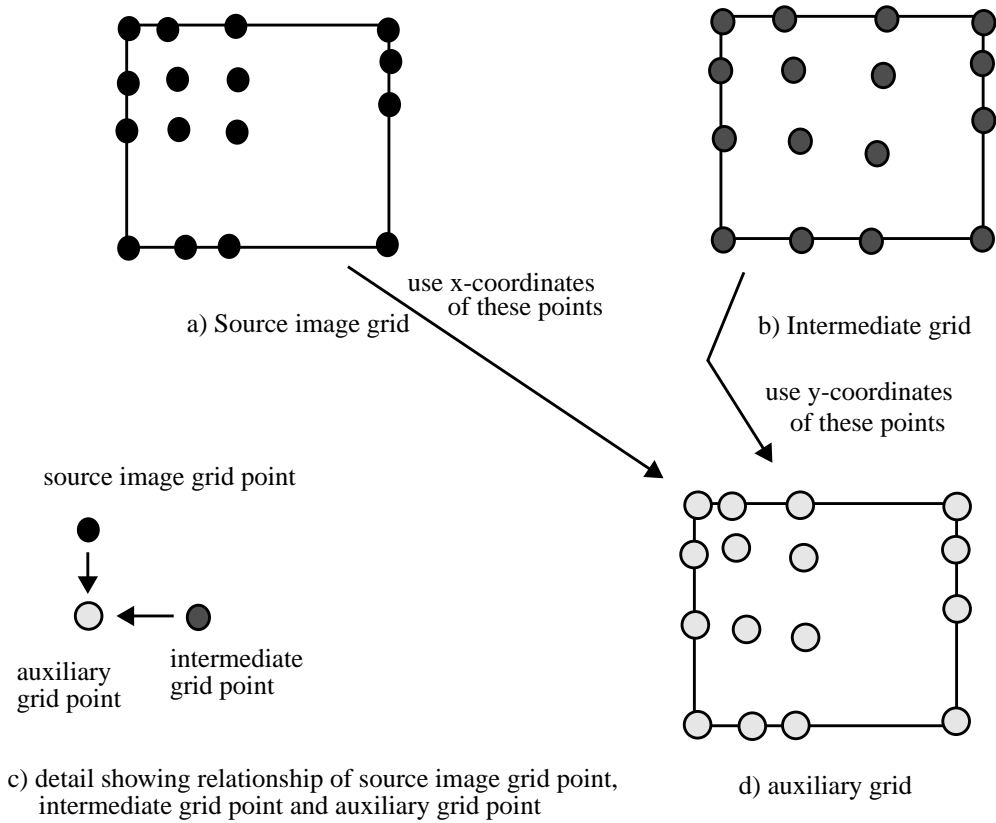


FIGURE 106. Formation of auxiliary grid for two-pass warping of source image to intermediate grid.

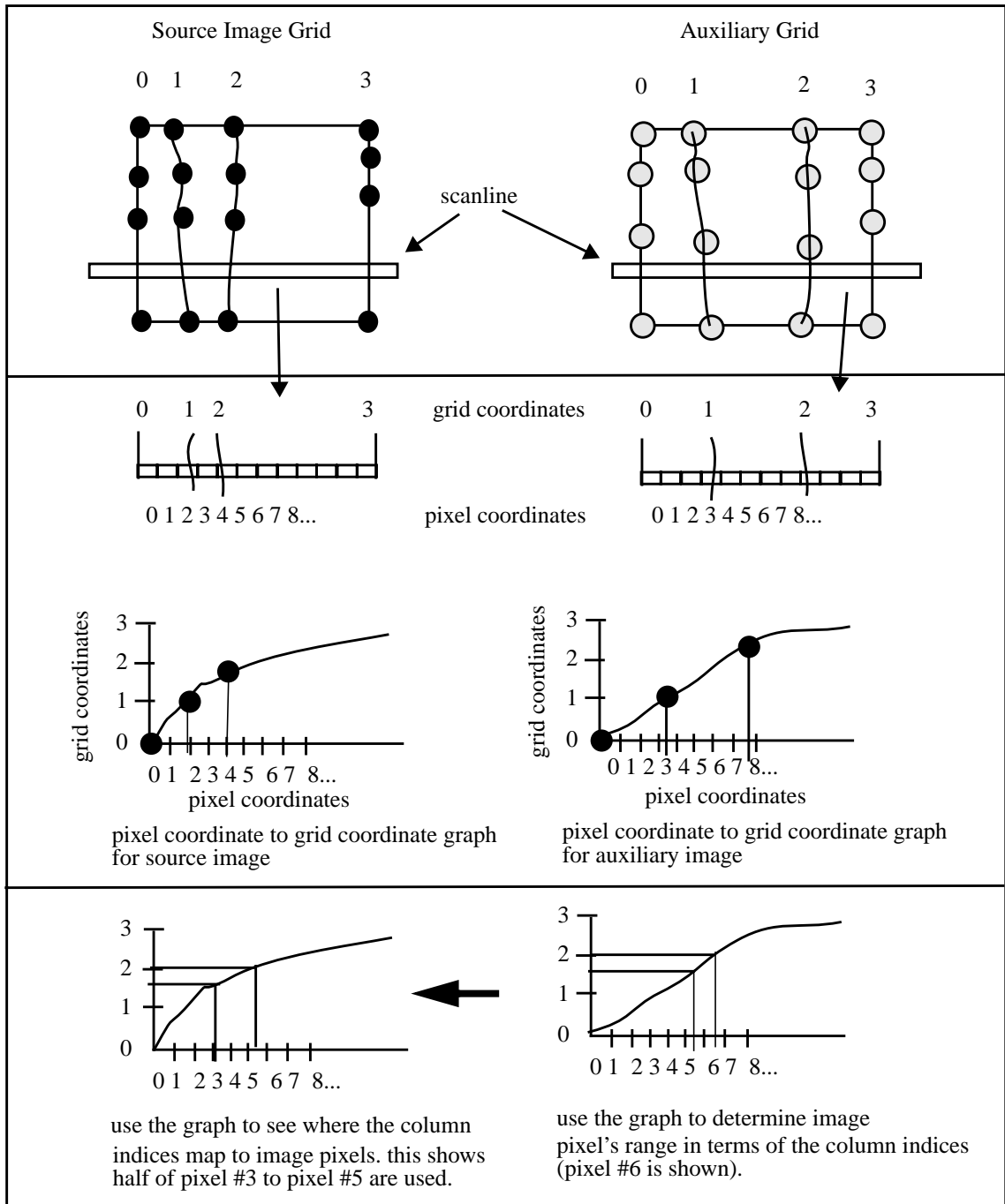
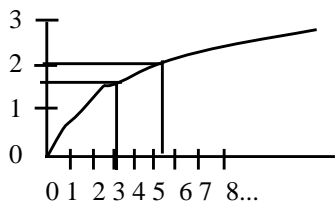
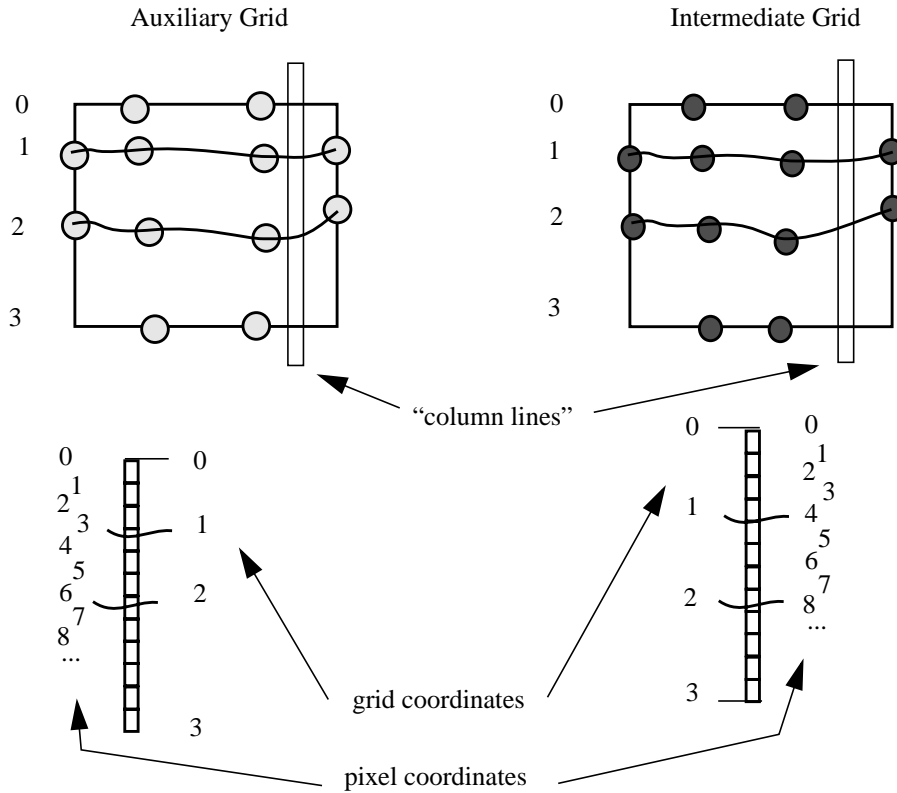
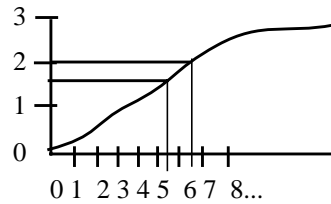


FIGURE 107. For given pixel in the auxiliary image, determine the range of pixel coordinates in the source image For example, pixel 6 of auxiliary grid maps to pixel coordinates 3.1 to 5.5 of image



Use row index coordinates to determine the pixel coordinates in auxiliary image.



For a given pixel in the intermediate image, determine the coordinates in terms of row indices.

FIGURE 108. Establishing the auxiliary pixel range for a pixel of the intermediate image. For example, pixel 6 of the intermediate grid maps to pixel coordinates 3.1 to 5.5 of the auxiliary image.

$$C[i][j] = \alpha \cdot C_1[i][j] + (1 - \alpha) \cdot C_2[i][j]$$

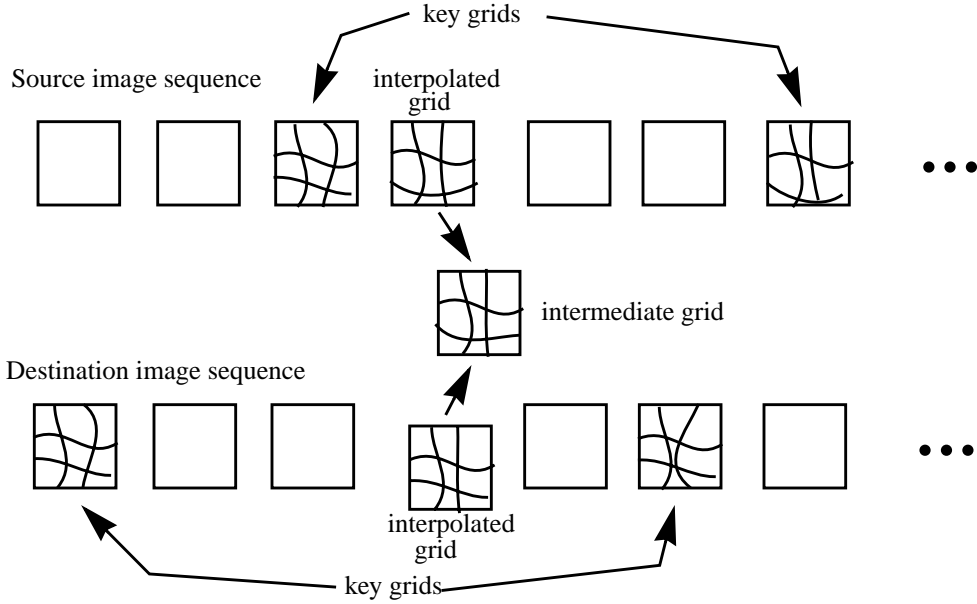
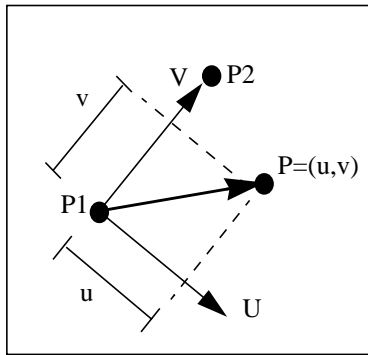


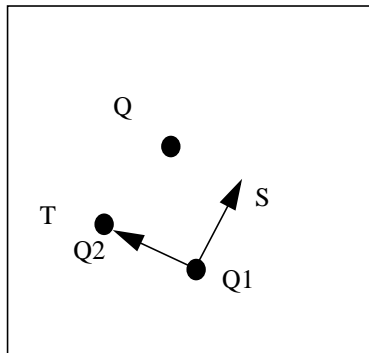
FIGURE 109. Morphing of animated sequences.



$$v = (P - P1) \cdot \frac{(P2 - P1)}{|P2 - P1|^2}$$

$$u = \left| (P - P1) \times \frac{(P2 - P1)}{|P2 - P1|^2} \right|$$

FIGURE 110. Local coordinate system of a feature in the intermediate image.



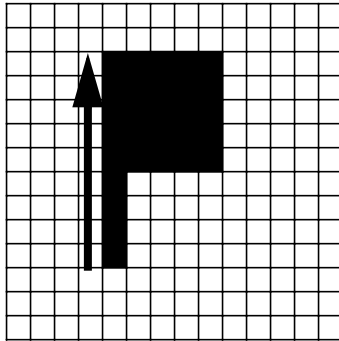
$$T = Q2 - Q1$$

$$S = (T_y, -T_x)$$

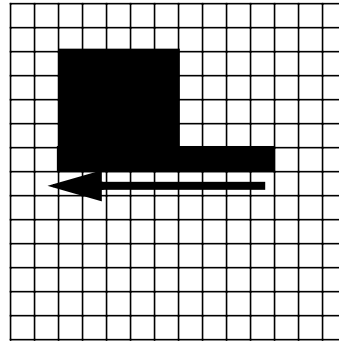
$$Q = Q1 + u \cdot S + v \cdot T$$

FIGURE 111. Relocating a point's position using local coordinates in the source image.

Source image and feature line

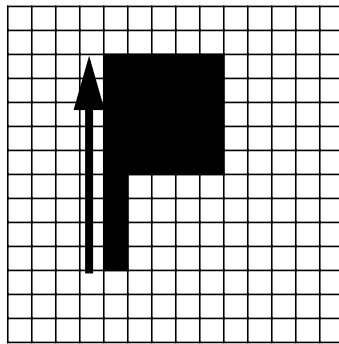


Intermediate feature line and resulting image

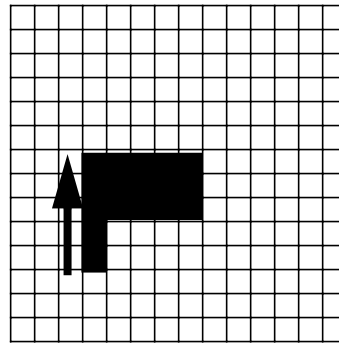


a) First example

Source image and feature line



Intermediate feature line and resulting image



b) Second example

FIGURE 112. Two examples of single feature line morphing.

$$w = \left(\frac{|Q2 - Q1|^p}{a + dist} \right)^b \tag{EQ 73}$$

```

/* for each pixel in the destination image */
for (i=0; i<xres; i++)
{
    for (j=0; j<yres; j++)
    {
        P.x = i; P.y = j;
        dispSumX = 0;
        dispSumY = 0;
        weightSum = 0;
        for (k=0; k<numFeatureLines; k++) {
            {
                P1 = featureLine[k].p1;
                P2 = featureLine[k].p2;
                Q1 = featureLine[k].q1;
                Q2 = featureLine[k].q2;

                V = formVector(P1,P2);
                Lv = getVectorLength(V);
                W = formVector(P1,P);
                Lw = getVectorLength(W);
                /* calculate (u,v) for P from line (P1,P2) */
                v = dot(W,V)/Lv*Lv;
                u = sqrt(Lw*Lw-Lv*Lv);
                T = formVector(Q1,Q2);
                S.x = T.y; S.y = -T.x;
                Q.x = Q1.x + u*S.x + v*T.x;
                Q.y = Q1.y + u*S.y + v*T.y;
                D = formVector(Q,P);
                WW = formVector(P2,P);
                Lww = getVectorLength(WW);
                dist = (v<0)? Lw: (v<1)? fabs(u): Lww
                weight = exp(length,p)/exp((a+dist),b);
                dispSum.x += D.x*weight;
                dispSum.y += D.y*weight;
                weightSum += weight;
            }
            Q.x = P.x + dispSum.x/weightSum;
            Q.y = P.y + dispSum.y/weightSum;
            si = (int)(Q.x+0.5);
            sj = (int)(Q.y+0.5);
            DestinationColorBuffer[i][j] = SourceColorBuffer[si][sj];
        }
    }
}

```

FIGURE 113. Feature line morphing code.

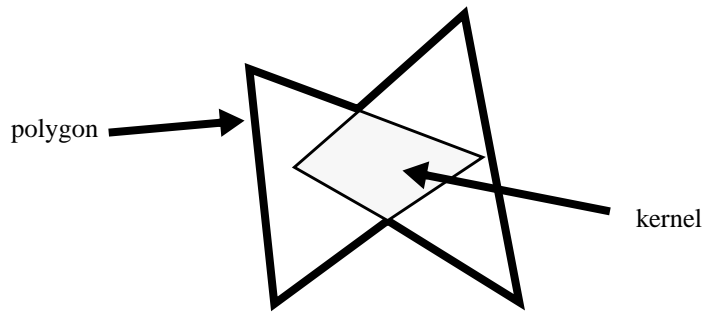
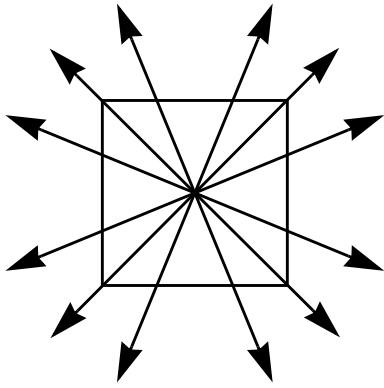
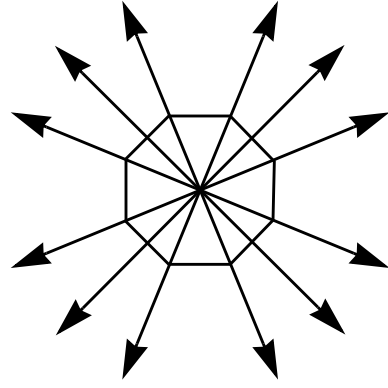


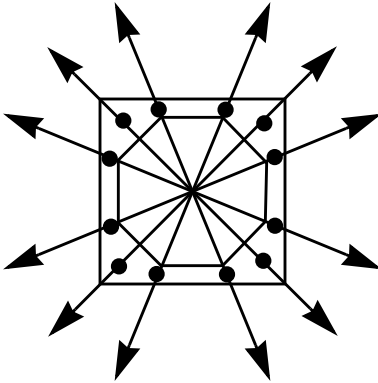
FIGURE 114. Star-shaped polygon and corresponding kernel



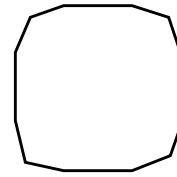
a) sampling Object 1 along rays



b) sampling Object 2 along rays



c) Points interpolated half-way between objects



d) resulting object

FIGURE 115. Star-shaped polyhedral shape interpolation

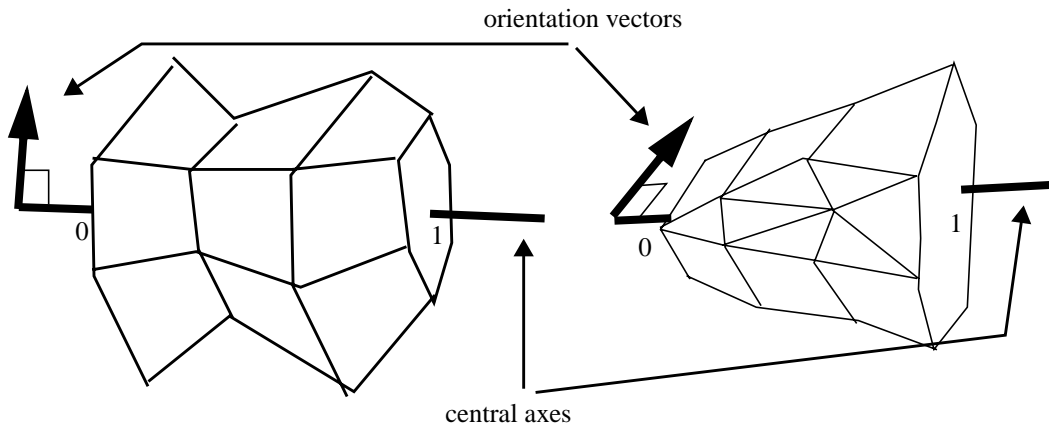


FIGURE 116. Coordinate system for axial slices.

).

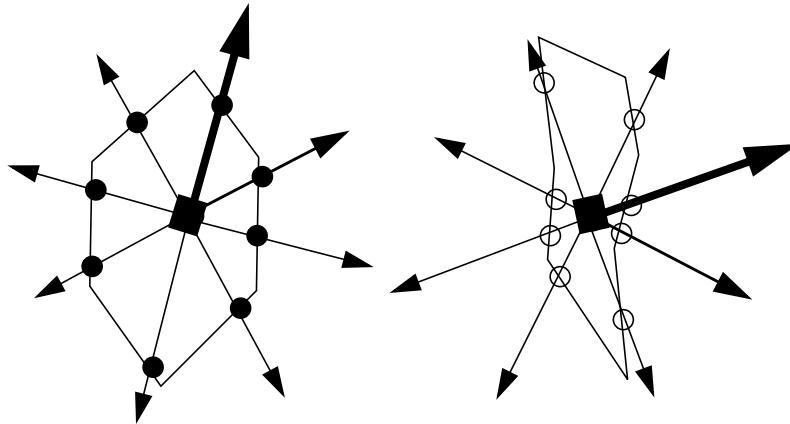
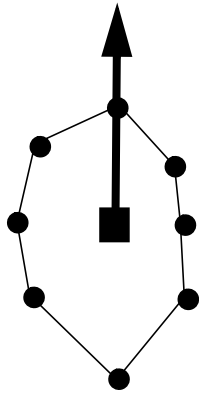
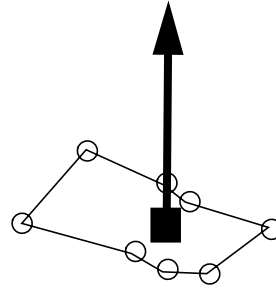


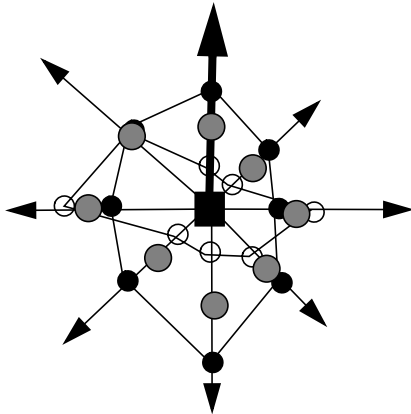
FIGURE 117. Projection lines for star-shaped polygons from objects in Figure 118.



a) slice from Object 1 showing reconstructed polygon.

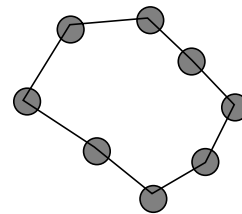


b) slice from Object 2 showing reconstructed polygon



c) Superimposed slices showing interpolated points

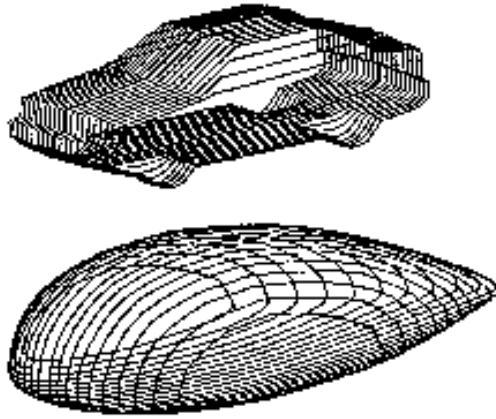
- - point from object 1
- - point from object 2
- - interpolated point



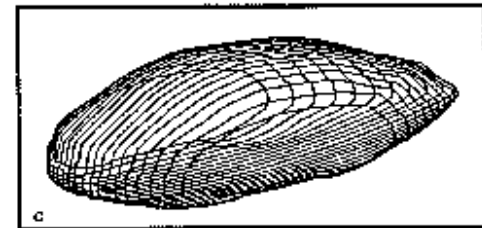
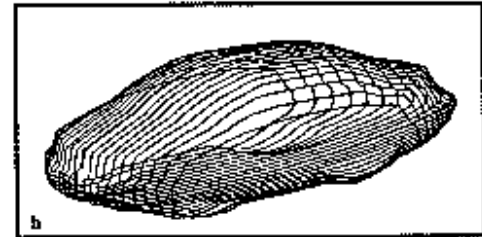
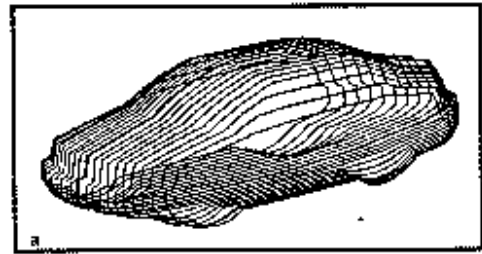
d) slice reconstructed from interpolated points

FIGURE 118. Interpolating slices taken from two objects along each respective central axis.

FIGURE 119. 3D shape interpolation from multiple 2D slices [12].



a) original shapes sliced into contours



b) interpolated shapes

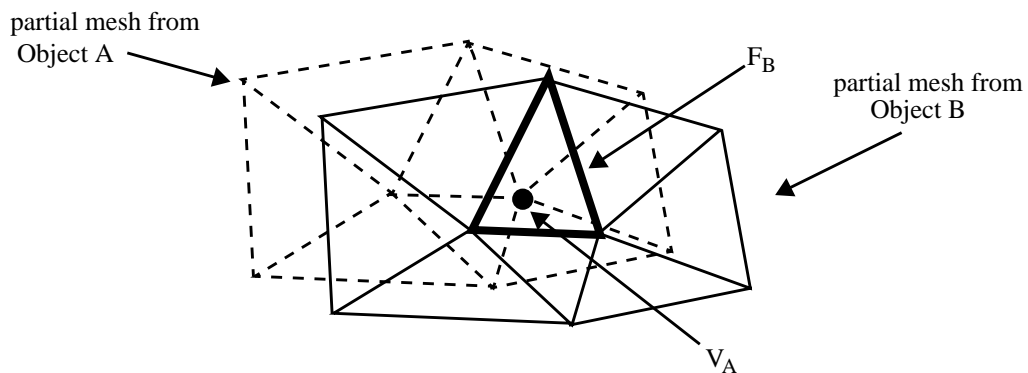


FIGURE 120. Locating initial vertex of Object A in face of Object B.

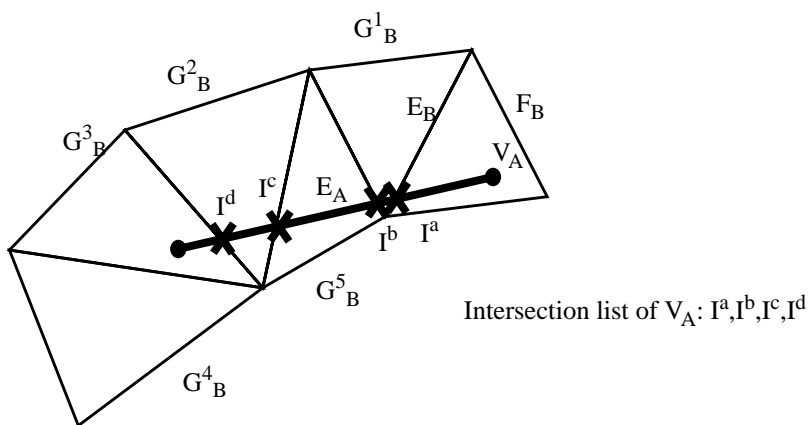


FIGURE 121. The intersection list for edge E_A

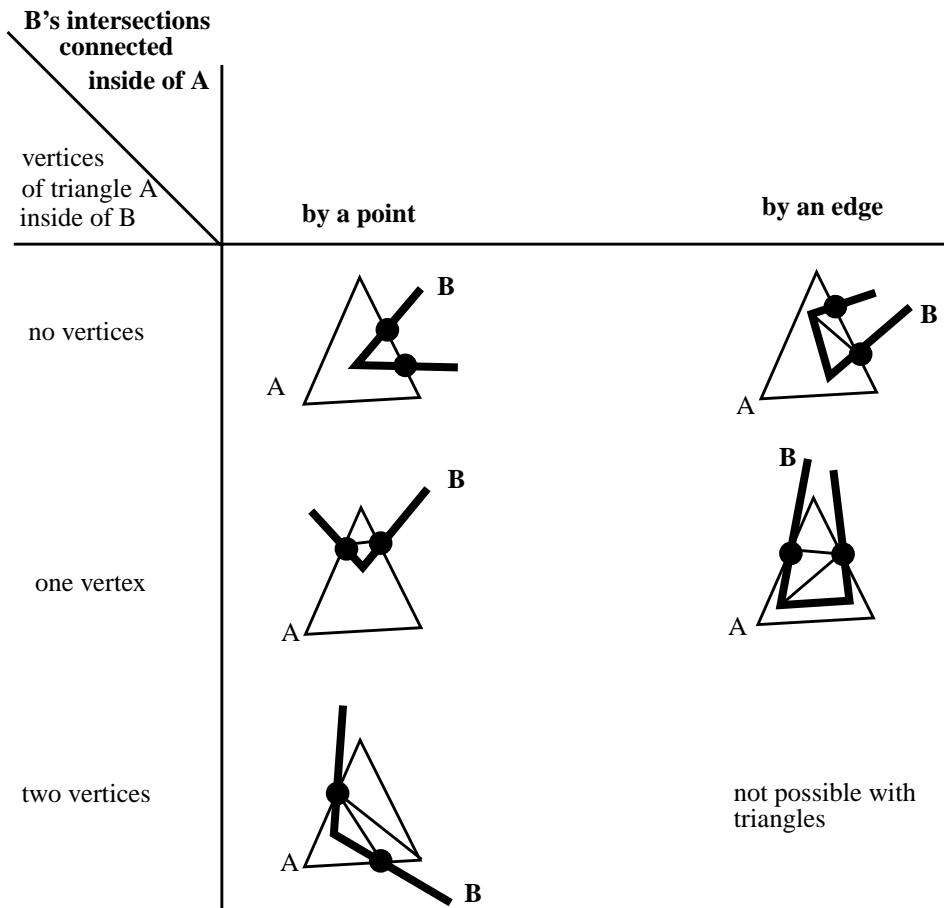


FIGURE 122. Configurations possible with overlapping triangles and possible triangulations.

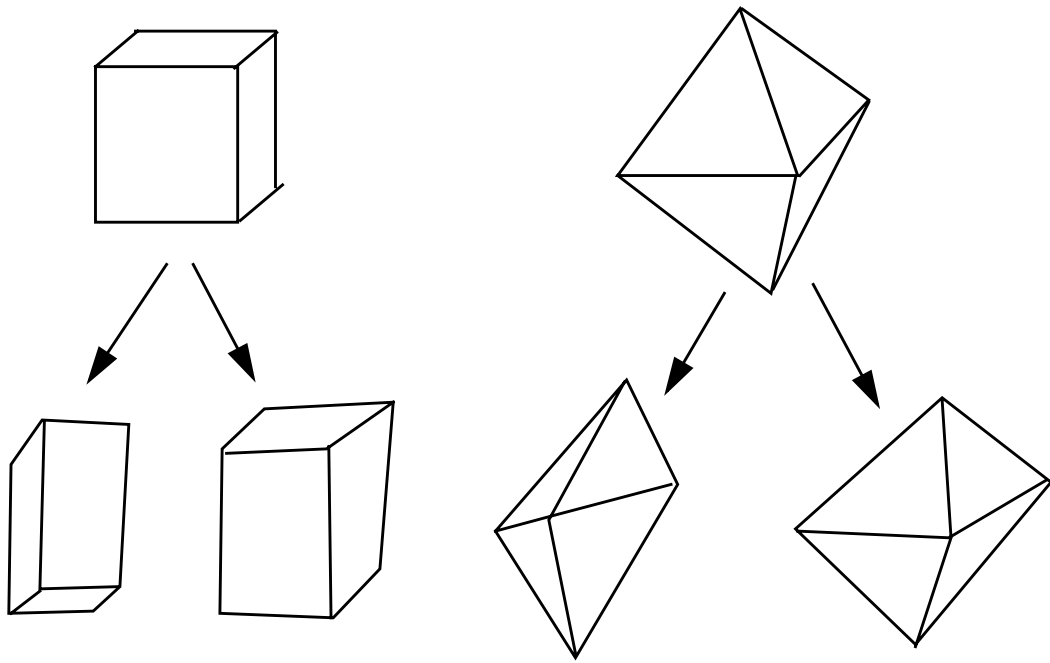
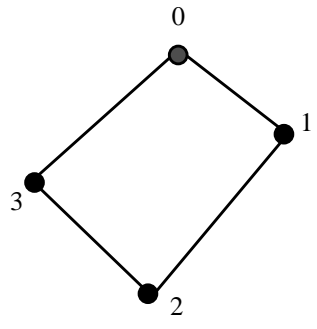
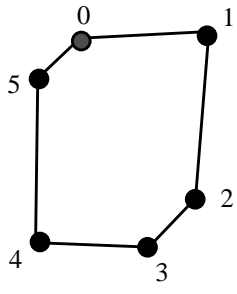


FIGURE 123. Splitting objects into initial front and back meshes.



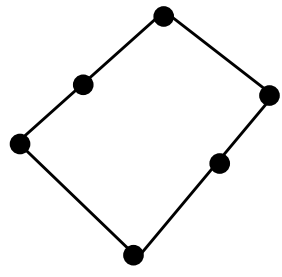
● - first vertex of boundary

normalized distances

0	0.00
1	0.15
2	0.20
3	0.25
4	0.40
5	0.70

normalized distances

0	0.00
1	0.30
2	0.55
3	0.70



boundary after adding additional vertices

FIGURE 124. Associating vertices of boundaries.

