Manual of Computer Graphics, 1st Edition. Program and PseudoCode Listings

April 2011

(Advise the author about missing or bad listings.)

Chapter 2

```
var srcWidth, destWidth: integer;
var srcPos=0, destPos=0, numerator=0: integer;
while(destPos < destWidth)
  dest[destPos]:=src[srcPos];
  destPos:=destPos+1;
  numerator:=numerator+srcWidth;
  while(numerator > destWidth)
    numerator:=numerator-destWidth;
    srcPos=:srcPos+1
  endwhile;
endwhile;
```

Figure 2.15. Bitmap Scaling Following Bresenham.

```
var srcWidth, destWidth, pixelFrac, num: integer;
var srcPos=0, destPos=0: integer;
pixelFrac:=destWidth;

while(destPos < destWidth)
  p:=0;
  num:=0;

    /* Handle whole pixels first */
  while(num+pixelFrac ≤ srcWidth)
    num:=num+pixelFrac;
    p:=p+pixelFrac × src[srcPos];
    srcPos:=srcPos+1
    pixelFrac:=destWidth;
  endwhile

  if(num<srcWidth)
    /* Partial pixel? */
    p:=p+(srcWidth-num) × src[srcPos];
    pixelFrac:=pixelFrac-(srcWidth-num);
  endif
  dest[destPos]:=p/srcWidth;
  destPos:=destPos+1;
endwhile;
```

Figure 2.19. Smooth Bitmap Scaling.

```
Initialization
srcWidth=13; destWidth=5;
pixelFrac=5; srcPos=destPos=0;

p:=0; num:=0; \* Iteration 1 *\
while(num+pixelFrac≤13)
 num:=0+5=5;
```

```
 p:=p+5×src[0];
 srcPos:=1;
 pixelFrac:=5;
 num:=5+5=10;
 p:=p+5×src[1];
 srcPos:=2
 pixelFrac:=5;
endwhile
if(10<13)
 p:=p+[13-10]×src[2];
 pixelFrac:=5-(13-10)=2;
endif
dest[0]:=(5src[0]+5src[1]+3src[2])/13;
destPos:=1;

p:=0; num:=0; \* Iteration 2 *\
while(num+pixelFrac≤13)
 num:=0+2=2;
 p:=p+2×src[2];
 srcPos:=3;
 pixelFrac:=5;
 num:=2+5=7;
 p:=p+5×src[3];
 srcPos:=4;
 pixelFrac:=5;
 num:=7+5=12;
 p:=p+5×src[4];
 srcPos:=5;
 pixelFrac:=5;
endwhile
if(12<13)
 p:=p+[13-12]×src[5];
 pixelFrac:=5-(13-12)=4;
endif
dest[1]:=(2src[2]+5src[3]+5src[4]+src[5])/13;
destPos:=2;

p:=0; num:=0; \* Iteration 3 *\
while(num+pixelFrac≤13)
 num:=0+4=4;
 p:=p+4×src[5];
 srcPos:=6;
 pixelFrac:=5;
 num:=4+5=9;
 p:=p+5×src[6];
 srcPos:=7;
 pixelFrac:=5;
endwhile
if(9<13)
 p:=p+[13-9]×src[7];
 pixelFrac:=5-(13-9)=1;
endif
dest[2]:=(4src[5]+5src[6]+4src[7])/13;
destPos:=3;

p:=0; num:=0; \* Iteration 4 *\
while(num+pixelFrac≤13)
 num:=0+1=1;
 p:=p+1×src[7];
 srcPos:=8;
 pixelFrac:=5;
 num:=1+5=6;
 p:=p+5×src[8];
 srcPos:=9;
 pixelFrac:=5;
 num:=6+5=11;
 p:=p+5×src[9];
 srcPos:=10;
 pixelFrac:=5;
endwhile
if(11<13)
 p:=p+[13-11]×src[10];
 pixelFrac:=5-(13-11)=3;
endif
```

```
dest[3]:=(src[7]+5src[8]+5src[9]+2src[10])/13;
destPos:=4;

p:=0; num:=0; \* Iteration 5 *\
while(num+pixelFrac≤13)
 num:=0+3=3;
 p:=p+3×src[10];
 srcPos:=11;
 pixelFrac:=5;
 num:=3+5=8;
 p:=p+5×src[11];
 srcPos:=12;
 pixelFrac:=5;
endwhile
if(8<13)
 p:=p+[13-8]×src[12];
 pixelFrac:=5-(13-8)=0;
endif
dest[3]:=(3src[10]+5src[11]+5src[12])/13;
destPos:=5;
```

Table 2.22. Smooth Bitmap Shrinking, 13 To 5 Example.

```
Proc line(source, y1, y2, destin, x1, x2);
var y, dx, dy, d: integer;
y:=y1;
dx:=x2-x1; dy:=y2-y1;
d:=2dy-dx;
for i:=x1 to x2 do
 dest(i):= source(y);
 while d ≥ 0 do
  y:=y+1;
  d:=d+dx;
 endwhile
 d:=d+2dy
endfor
```

Figure 2.23. Stretching An Array.

```
if ((B ≠ H) and (D ≠ F)) then
  E0:=if (D = B) then D else E endif;
  E1:=if (B = F) then F else E endif;
  E2:=if (D = H) then D else E endif;
  E3:=if (H = F) then F else E endif; else
  E0:=E1:=E2:=E3:=E
endif
```

Figure 2.25. The Scale2 Algorithm.

```
if ((B ≠ H) and (D ≠ F)) then
  E0:=if (D = B) then D else E endif;
  E1:=if ((D = B) and (E ≠ C)) or ((B = F) and (E ≠ A)) then B else E endif;
  E2:=if (B = F) then F else E endif;
  E3:=if ((D = B) and (E ≠ G)) or ((D = H) and (E ≠ A)) then D else E endif;
  E4:=E;
  E5:=if ((B = F) and (E ≠ I)) or ((H = F) and (E ≠ C)) then F else E endif;
  E6:=if (D = H) then D else E endif;
  E7:=if ((D = H) and (E ≠ I)) or ((H = F) and (E ≠ G)) then H else E endif;
  E8:=if (H = F) then F else E endif; else
```

```
  E0:=E1:=E2:=E3:=E4:=E5:=E6:=E7:=E8:=E
endif
```

Figure 2.26. The Scale3 Algorithm.

```
E0:=E1:=E2:=E3:=E;
if (A = B = D) then E0:=A endif;
if (B = C = F) then E1:=C endif;
if (D = G = H) then E2:=G endif;
if (F = I = H) then E3:=I endif;
```

Figure 2.28. The Eagle Scaling Algorithm.

```
Clear[Nh,P,U,W];
Nh={{-4.5,13.5,-13.5,4.5},{9,-22.5,18,-4.5},
 {-5.5,9,-4.5,1},{1,0,0,0}};
P={{p33,p32,p31,p30},{p23,p22,p21,p20},
 {p13,p12,p11,p10},{p03,p02,p01,p00}};
U={u^3,u^2,u,1};
W={w^3,w^2,w,1};
u:=0.5;
w:=0.5;
Expand[U.Nh.P.Transpose[Nh].W]
```

Code in Section 2.12.3.

```
Clear[Nh,p,pnts,U,W];
p00={0,0,0}; p10={1,0,1}; p20={2,0,1}; p30={3,0,0};
p01={0,1,1}; p11={1,1,2}; p21={2,1,2}; p31={3,1,1};
p02={0,2,1}; p12={1,2,2}; p22={2,2,2}; p32={3,2,1};
p03={0,3,0}; p13={1,3,1}; p23={2,3,1}; p33={3,3,0};
Nh={{-4.5,13.5,-13.5,4.5},{9,-22.5,18,-4.5},
 {-5.5,9,-4.5,1},{1,0,0,0}};
pnts={{p33,p32,p31,p30},{p23,p22,p21,p20},
 {p13,p12,p11,p10},{p03,p02,p01,p00}};
U[u_]:={u^3,u^2,u,1};   W[w_]:={w^3,w^2,w,1};
(* prt [i] extracts component i from the 3rd dimen of P *)
prt[i_]:=pnts[[Range[1,4],Range[1,4],i]];
p[u_,w_]:={U[u].Nh.prt[1].Transpose[Nh].W[w],
 U[u].Nh.prt[2].Transpose[Nh].W[w], \
 U[u].Nh.prt[3].Transpose[Nh].W[w]};
 g1=ParametricPlot3D[p[u,w], {u,0,1},{w,0,1},
 Compiled->False, DisplayFunction->Identity];
g2=Graphics3D[{AbsolutePointSize[2],
 Table[Point[pnts[[i,j]]],{i,1,4},{j,1,4}]}];
Show[g1,g2, ViewPoint->{-2.576, -1.365, 1.718}]
```

Code For Figure Ans.1.

```
diff:=round(1000/scale);
accum_diff:=0;
j:=1;
for y:=1 to N do
Q[x,y]:=P[i,j];
a:=⌊accum_diff/1000⌋;
accum_diff:=accum_diff+diff;
b:=⌊accum_diff/1000⌋;
j:=j+(b-a);
```

```
endfor;
```

Figure 2.32. Scaling One Scan Line.

```
d1:=|x_5-x_2|;
d2:=|x_5-x_4|;
d3:=|x_5-x_1|;
d4:=|x_5-(x_2+x_4)/2|;
min:=minimum(d1,d2,d3,d4);
if(min=d1) then y_1:=(x_5+x_2)/2 else
if(min=d2) then y_1:=(x_5+x_4)/2 else
if(min=d3) then y_1:=(x_5+x_1)/2 else
if(min=d4) then y_1:=x_5/2+x_2/4+x_4/4;
```

Figure 2.40. A Simpler Scaling Algorithm.

```
t=Pi/4;
Brot[x_,y_]:={x Cos[t]+y Sin[t],-x Sin[t]+y Cos[t]};
Do[Print[x,",",y,N[Brot[x,y],1]], {x,-4,4},{y,-4,4}]
```

Code in Section 2.15.

```
procedure xshear(α,r,c);
 for i:=-r to r do
  skew:=i*α;
  int:=floor(shear);
  f:=frac(shear);
  PrevLeft:=0;
  for j:=-c to c do
   p:=Sbitmap[i,j];
   LeftPart:=p*f;
   Dbitmap[i,j+int]:=p-LeftPart+PrevLeft;
   PrevLeft:=LeftPart;
  endfor;
 Dbitmap[i,int]:=PrevLeft;
 endfor;
end;
```

Figure 2.48. Shearing in the x Direction.

```
Remove["Global`*"];
L = Table[{i+RandomReal[{-0.15, 0.15}],
 j+RandomReal[{-0.15,0.15}]},{i,0,9},{j,0,9}];
P1=ListLinePlot[L, Axes->False];
P2=ListLinePlot[Transpose[L], Axes->False];
Show[{P1, P2}, AspectRatio->Automatic]
```

Figure 2.52. Square and Triangular Grids For Arbitrary Image Deformation.

```
for i := 1 to m do
 for j := 1 to n do
  begin
  B[i,j]:=SearchPalette(A[i,j]);
```

```
  err := A[i,j] − B[i,j];
  A[i,j+1] := A[i,j+1] + err ∗ p1;
  A[i+1,j−1] := A[i+1,j−1] + err ∗ p2;
  A[i+1,j] := A[i+1,j] + err ∗ p3;
  A[i+1,j+1] := A[i+1,j+1] + err ∗ p4;
  end.

for i := 1 to m do
 for j := 1 to n do
  begin
  if A[i,j] < 0.5 then B[i,j] := 0
    else B[i,j] := 1;
  err := A[i,j] − B[i,j];
  A[i,j+1] := A[i,j+1] + err ∗ p1;
  A[i+1,j−1] := A[i+1,j−1] + err ∗ p2;
  A[i+1,j] := A[i+1,j] + err ∗ p3;
  A[i+1,j+1] := A[i+1,j+1] + err ∗ p4;
  end.
```

Figure 2.78. Diffusion Dither Algorithm.

```
for k := 0 to 63 do
 for all (i,j) of class k do
  begin
  if A[i,j] < .5 then B[i,j] := 0 else B[i,j] := 1;
  err := A[i,j] − B[i,j];
  Distribute(err,i,j,k);
  end.

procedure Distribute(err,i,j,k);
 w := 0;
 for all neighbors A[u,v] of A[i,j] do
  if class(u,v) > k then w := w+weight(u − i,v − j);
 if w > 0 then for all neighbors A[u,v] of A[i,j] do
  if class(u,v) > k then A[u,v] := A[u,v] + err×weight(u − i,v − j)/w;
end;
```

Figure 2.80. The Dot Diffusion Algorithm.

```
(* Stippling an image *)
ar=Import["A1965.jpg"] (* Input grayscale image *)
Shallow[InputForm[ar]];
d=ar[[1, 1]]; (* convert image to an array of pixels *)
btmp=d[[All,All,1]]; (* Leave 1 gray value per pixel *)
{row,col}=Dimensions[btmp]
stp=Table[0, {i,1,row}, {j,1,col}]; (* Init stp to zeros *)
Do[If[btmp[[i,j]]<RandomInteger[150], stp[[row+1-i,j]]=1],
 {i,1,row}, {j,1,col}];
tot=Total[stp, {1,3}] (* Total elements of stp *)
N[tot/(row col)] (* Percentage of black dots *)
ArrayPlot[stp]
```

Figure 2.83. Original Grayscale and Two Stippled Images.

```
(* Place random points in a circle *)
```

```
n=300; R=10;
theta=Table[Random[Real, {0,2 Pi}], {n}];
r=Table[Random[Real, {0,R}], {n}]; (* uniform distribution *)
P=Point[
   Table[{r[[i]] Cos[theta[[i]]], r[[i]] Sin[theta[[i]]]}, {i,1,n}]];
(* points are denser toward the center *)
Show[
 Graphics[{Green, P}], Graphics[{Blue, Circle[{0, 0}, 10]}],
 Graphics[{Red, Point[{0, 0}]}], AspectRatio->1]
 (* Now give r a ramp distribution *)
r=Table[Max[Random[Real, {0,R}], Random[Real, {0,R}]], {n}];
P = Point[
   Table[{r[[i]] Cos[theta[[i]]], r[[i]] Sin[theta[[i]]]}, {i,1,n}]];
(* points are uniformaly distributed *)
Show[Graphics[{Green, P}],
 Graphics[{Blue, Circle[{0,0},10]}], Graphics[{Red, Point[{0,0}]}],
  AspectRatio->1]
```

Figure 2.87. Two Distributions of Random Points in a Circle.

```
(* Gaussian Kernel (normalized) *)
sigma=0.84089642;
sigmat=2.sigma^2;
cc=1/sigmat Pi;
gausskernel[x_, y_]:=cc E^(-(x^2+y^2)/sigmat);
GC=Table[gausskernel[x,y], {x,-3,3}, {y,-3,3}];
GC=GC/Total[Flatten[GC]] (* Normalize *)
Plot3D[gausskernel[x,y], {x,-3,3}, {y,-3,3}, PlotRange->All]
```

Figure 2.90. Gaussian Kernel.

Chapter 3

```
var a, b, x, y, x1, x2, y1, y2: real;
a:=(y2-y1)/(x2-x1);
b:=y1-a*x1;
x:=x1;
repeat
y:=a*x+b;
point(round(x),round(y));
x:=x+1;
until x>x2;
```

Figure 3.1. Scan Convert $y = ax + b$.

```
void Midpoint(int a1,int b1, int a2, int b2)
{
int midx,midy;
midx=(a1+a2)/2; midy=(b1+b2)/2;
putpixel(midx,midy,Color); /* Turbo C */
if(abs(a1-midx)>1 || abs(b1-midy)>1) Midpoint(a1,b1,midx,midy);
if(abs(midx-a2)>1 || abs(midy-b2)>1) Midpoint(midx,midy,a2,b2);
}
```

Pseudocode for Midpoint Subdivision

```
var x, x1, y1, x2, y2: integer; a, y: real;
a:=(y2-y1)/(x2-x1);
```

```
x:=x1; y:=y1;
repeat
point(x,round(y));
x:=x+1; y:=y+a;
until x>x2;
```

Code for simple DDA

```
var Δx,Δy,L:integer; x,y,a,G,H:real;
Δx:=x2-x1; Δy:=y2-y1;
x:=x1; y:=y1;
if Δx>Δy then G:=1; H:=a;
else G:=1/a; H:=1 endif;
for L:=1 to max(Δx,Δy)+1 do
point(round(x),round(y));
x:=x+G; y:=y+H;
endfor;
```

Code for Simple DDA

```
procedure SimpleDDA(x1,y1,x2,y2: integer);
begin Δx,Δy,length,i:integer; x,y,x_incr,y_incr:real;
Δx:=x2-x1; Δy:=y2-y1;
length:=max(abs(Δx), abs(Δy));
x_incr:=Δx/length; y_incr:=Δy/length;
x:=x1; y:=y1;
for i:=1 to length+1 do
point(round(x),round(y));
x:=x+x_incr; y:=y+y_incr;
endfor;
end;
```

A Variation

```
procedure SymmDDA(x1,y1,x2,y2: integer);
calculate eps;
xIncr:=eps*Δx;
yIncr:=eps*Δy;
x:=x1+.5; y:=y1+.5;
repeat
Plot(trunc(x),trunc(y));
x:=x+xIncr; y:=y+yIncr;
until x=x2 or y=y2;
end;
```

Symmetrical DDA

```
var x1,y1,x2,y2,Δx,Δy: integer;
Δx:=x2-x1; Δy:=y2-y1;
Err:=0;
repeat
plot(x1,y1);
if Err>0 then
 x1:=x1+1;
```

```
 Err:=Err-Δy
else
 y1:=y1+1;
 Err:=Err+Δx
endif;
until x1>=x2 and y1>=y2;
```

Quadrantal DDA

```
var dx, dy, dxdy, D: integer;
dxdy:=dy-dx, D:=0;
x:=x1; y:=y1;
repeat
  pixel(x,y);
  if d>0 then y:=y+1; D:=D+dxdy
   else D:=D+dy
  endif;
  x:=x+1;
until x=x2;
```

Bresenham's Method

```
var x,y,dxy,dy,d: integer;
y:=y1;
Δx=x2-x1; Δy=y2-y1;
dy:=2Δy; dxy:=2(Δy − Δx);
d:=2Δy − Δx;
for x:=x1 to x2 do
 pixel(x,y);
 if d< 0 then d:=d+dy;
  else d:=d+dxy; y:=y+1
 endif
endfor
```

Bresenham's Line Method

```
bresenham(int x1,y1,x2,y2)
{int y=y1, dx=x2-x1, dy=y2-y1;
 int d=2*dy-dx;
 for (int x=x1; x<=x2; x++)
    {
   pixel(x,y);
   if(d<0) d+=2*dy;
   else{y++; d+=2*(dy-dx);
       }
     }
}
```

Bresenham's Line Method

```
procedure dbstep(x1,y1, x2,y2);
dx=x2-x1; dy=y2-y1;
x:=x1; y:=y1;
if dy/dx>.5 (high slope)
```

```
then
 while x≤x2 do
  if D<0 then pattern(5) else pattern(4);
 endwhile;
else        (low slope)
 while x≤x2 do
  if D<0 then pattern(1) else pattern(5);
 endwhile;
endif;
procedure pattern(patt: integer);
case patt of
 1: pixel(x+1,y); pixel(x+2,y);
 2: pixel(x+1,y); pixel(x+2,y+1); y:=y+1;
 3: pixel(x+1,y+1); pixel(x+2,y+1); y:=y+1;
 4: pixel(x+1,y+1); pixel(x+2,y+2); y:=y+2;
end (case)
x:=x+2;
end;
```

Double-Step DDA

```
procedure bestfit(x1,y1,x2,y2: integer);
var str1,str2: 0..1;
    x,y,i: integer;
    done: boolean;
begin
 done:=false;
 str1:='0'; str2:='1';
 y:=y2-y1;
 x:=(x2-x1)-y;
 repeat
  case
   x>y: str2:=rev(str2)+str1; x:=x-y;
   x=y: done:=true;
   x<y: str1:=rev(str1)+str2; y:=y-x;
  end; (* case *)
 until done;
 str1:=rev(str2)+str1;
 (* or str1:=rev(str1)+str2 *)
 duplicate str1 x times;
 x:=x1; y:=y1; pixel(x,y);
 for i:=1 to len(str1) do
  begin (* actual drawing *)
  x:=x+1;
  if substring(str1,i,i)='1' then y:=y+1
  pixel(x,y);
 end; (* for *)
end.
```

Figure 3.12. The Best-Fit DDA Method.

```
for x:=0 to R step eps do
y:=sqrt(R*R-x*x);
plot(x,y); plot(-x,y);
```

```
plot(x,-y); plot(-x,-y);
end;
```

Obvious Method for Scan-Converting Circles

```
for theta:=0 to pi/2 step eps do
x:=R*cos(theta); y:=R*sin(theta);
plot(x,y); plot(-x,y);
plot(x,-y); plot(-x,-y);
end;
```

Obvious Method for Scan-Converting Circles

```
input(n,delta,a,b,R);
xk:=R; yk:=0;
dcos:=Cos(delta); dsin:=Sin(delta);
for k:=0 to n-1 do
xn:=xk*dcos-yk*dsin;
yn:=xk*dsin+yk*dcos;
xk:=xn; yk:=yn;
pixel(round(xn)+a,round(yn)+b);
end;
```

Circle in Polar Coordinates

```
Clear[L];
n=18; delta=5 Degree; R=1;
xk=R; yk=0;
dcos=Cos[delta]//N; dsin=Sin[delta]//N;
L={};
Do[xn=xk dcos-yk dsin; yn=xk dsin+yk dcos;
xk=xn; yk=yn; L=Append[L,{xn,yn}], {k,0,n-1}];
L
ListPlot[L, Prolog->AbsolutePointSize[3],
 AspectRatio->Automatic]
```

Code for Figure Ans.9

```
x:=0; y:=R;
while x<y do
 Plot(x,y);
 .
 .
 if d>0 then
 .
 .
 y:=y-1;
 else
 ...
 endif;
 x:=x+1;
endwhile;
```

Figure 3.17. Main Loop of Bresenham's Circle Algorithm.

```
procedure Bresenham(R);
x:=0; y:=R; d:=3-2*R;
while x<y do
Plot8(x,y);
if d>0 then
 d:=d+4*(x-y)+10;
 y:=y-1;
else
 d:=d+4*x+6;
endif;
x:=x+1;
endwhile;
if x=y then Plot8(x,y)
end; {Bresenham}

procedure Plot8(x,y);
Plot(x,y);
Plot(-x,-y);
Plot(-x,y);
Plot(x,-y);
Plot(y,x);
Plot(-y,-x);
Plot(-y,x);
Plot(y,-x);
end; {Plot8}
```

Figure 3.18. Bresenham's Circle Algorithm.

```
DCS (int r){
int i = 0,  j = r,  s = 0,  w = r - 1;
int l = w << 1;
while (j >= i){
  do{Plot8(i,j);
      s = s + i;
      i++;
      s = s + i;}
while (s <= w);
w = w + l;
l = l - 2;
j--;
              }
          }
```

DCS Circle Method

```
Push the seed pixel onto the stack
  While the stack is not empty
    Pop a pixel P from the stack
    Set P to color f
    For each of the four (or eight) nearest neighbors of P,
     If any is a boundary pixel or is already painted f, then disregard it
     else push it onto the stack.
```

Polygon Seed Fill

```
var x,x1,x2: integer; a,y: real;
a:=(y2-y1)/(x2-x1);
```

```
x:=x1; y:=y1;
repeat
pixel(x,trunc(y));
x:=x+1; y:=y+a;
until x>x2;

var x,x1,x2,numgray: integer; a,b,y:real;
calculate a and b;
x:=x1; y:=y1;
repeat
d=y-trunc(y);
gray1=(1-d)*numgray;
gray2=d*numgray;
pixel(x,trunc(y),gray1);
pixel(x,trunc(y)+1,gray2);
x:=x+1; y:=y+a;
until x>x2;
```

Figure 3.36. Simple DDA (a) Aliased, (b) Antialiased.

```
var x1,y1,x2,y2,maxgray,shft,V,v: integer;
shft:=16-5;
V:=0;
v:=(y2-y1)<<16 /(x2-x1);
repeat
companion:=V>>shft;
main:=companion xor 31;
pixel(x1,y1,main);
pixel(x1,y1+1,companion);
pixel(x2,y2,main);
pixel(x2,y2+1,companion);
VT:=V;
V:= V + v;
if  V < VT then y1:=y1+1; y2:=y2-1;
x1:=x1+1; x2:=x2-1;
until x2>x1;
```

Figure 3.38. Symmetric Wu Algorithm.

Chapter 4

```
d2r=Pi/180;
Table[Round[N[16384*Sin[i*d2r]]], {i,0,90}]
```

Fast Rotations

```
t14=2^14;
Print["(x*=",(8192-(2 14189.))/t14,",y*=",(14189.+(2 8192))/t14,")"]
Print["(x*=",Cos[60 Degree]-2. Sin[60 Degree],
 ",y*=",Sin[60 Degree]+2. Cos[60 Degree], ")"]

t14=2^14;
Print["(x*=",(2845.-(2 16135.))/t14,",y*=",(16135.+(2 2845.))/t14,
  ")"]
Print["(x*=",Cos[80 Degree]-2. Sin[80 Degree],
 ",y*=",Sin[80 Degree]+2. Cos[80 Degree], ")"]
```

Codes (in an answer) for fast rotation

```
t=Table[ArcTan[2.^{-i}], {i,0,15}]; (* arctans in radians *)
d=1; x=2.1; y=0.34; z=46. Degree;
Do[{Print[i,", ",x,", ",y,", ",z,", ",d],
 xn=x+y d 2^{-i}, yn=y-x d 2^{-i},
 zn=z-d t[[i+1]], d=Sign[zn], x=xn, y=yn, z=zn}, {i,0,14}]
Print[0.60725x,", ",0.60725y]
```

Figure 4.15. Code for CORDIC Rotations.

```
tm=Sqrt[x^2+y^2];
a={{x/tm,-y/tm,0},{y/tm,x/tm,0},{0,0,1}};
b={{z,0,Sqrt[1-z^2]},{0,1,0},{-Sqrt[1-z^2],0,z}};
c={{Cos[t],-Sin[t],0},{Sin[t],Cos[t],0},{0,0,1}};
FullSimplify[a.b.c.Transpose[b].Transpose[a] /. x^2+y^2->1-z^2]
```

Figure 4.29. Code for a General Rotation.

```
n=3;
A=[.5774,-.5774,-.5774; .5774,.7886,-.2115; .5774,-.2115,.7886]
% Rotation from 1,1,1 to x-axis
Q=eye(n);
for j=1:n-1,
  for i=n:-1:j+1,
    T=eye(n);
    D=sqrt(A(j,j)^2+A(i,j)^2);
    cos=A(j,j)/D; sin=A(i,j)/D;
    T(j,j)=cos; T(j,i)=sin; T(i,j)=-sin; T(i,i)=cos; T
    A=T*A;
Q=Q*T';
  end;
end;
Q
A
```

Figure 4.30. Computing Three Givens Matrices.

```
T1=[0.7071,0,0.7071; 0,1,0; -0.7071,0,0.7071];
T2=[0.8165,0.5774,0; -0.5774,0.8165,0; 0,0,1];
T3=[1,0,0; 0,0.9660,0.2587; 0,-0.2587,0.9660];
p=[1;1;1];
a=T1*p
b=T2*a
c=T3*b
```

Figure 4.31. Rotating Point $(1,1,1)$ to the x Axis.

Chapter 6

```
Clear[r,R,ta];
{R Cos[t],R Sin[t],R}.{{1,0,0},{0,Cos[ta],-Sin[ta]},{0,Sin[ta],Cos[ta]}}
{R Cos[t],R Cos[ta] Sin[t]+R Sin[ta],R Cos[ta]-R Sin[t] Sin[ta],1}.
 {{1,0,0,0},{0,1,0,0},{0,0,0,r},{0,0,0,1}}
R=1; r=.5; ta=45 Degree;
ParametricPlot[{R Cos[t]/(1+r R(Cos[ta]-Sin[t]Sin[ta])),
 R(Cos[ta]Sin[t]+Sin[ta])/(1+r R(Cos[ta]-Sin[t]Sin[ta]))}, {t,0,2Pi}]
```

Code to Experiment with the curve of Equation (6.2).

```
k = 3.; r = 1/k;
{a, b, c} = {0, 0, -k}; {d, e, f} = Normalize[{-1, 1, k}]
T = {{(e^2 + f + f^2)/(1 + f), -d e/(1 + f), 0, d r},
 {-d e/(1 + f), (d^2 + f + f^2)/(1 + f), 0, e r},
 {-d, -e, 0, f r},
 {(c d + b d e - a e^2 - a f + c d f - a f^2)/(1 + f),
 (-b d^2 + c e + a d e - b f + c e f - b f^2)/(1 + f),
 0, -(a d + b e + c f)  r}};
{-1, 1, 0, 1}.T

k = 3.; r = 1/k;
{a, b, c} = {0, 2k, -k}; {d, e, f} = Normalize[{0, -1, -1}]
T = {{(e^2 + f + f^2)/(1 + f), -d e/(1 + f), 0, d r},
 {-d e/(1 + f), (d^2 + f + f^2)/(1 + f), 0, e r},
 {-d, -e, 0, f r},
 {(c d + b d e - a e^2 - a f + c d f - a f^2)/(1 + f),
 (-b d^2 + c e + a d e - b f + c e f - b f^2)/(1 + f),
 0, -(a d + b e + c f)  r}};
{0,0,-4k,1}.T
```

Computes the Normalized Components of D.

```
{a,b,c}={0,1.,0}; {d,e,f}=Normalize[{0,1,1}]
T = {{(e^2 + f + f^2)/(1 + f), -d e/(1 + f), 0, d r},
 {-d e/(1 + f), (d^2 + f + f^2)/(1 + f), 0, e r},
 {-d, -e, 0, f r},
 {(c d + b d e - a e^2 - a f + c d f - a f^2)/(1 + f),
 (-b d^2 + c e + a d e - b f + c e f - b f^2)/(1 + f),
 0, -(a d + b e + c f)  r}};
{0,1,10,1}.T
```

(In an exercise) Code for Further Experimentation.

```
(* code to check matrix T_g for the case 1 + f = 0 *)
r = 1/k; {a, b, c} = {0, 0, -k};
T = {{-1, 0, 0, 0}, {0, -1, 0, 0}, {0, 0, 0, -r}, {a, b, 0, c r}};
{x, y, z, 1}.T
```

Code to Test Matrix (6.16).

```
a = 1/Sqrt[2]; h = a; i = a;
T = {{i, h, 0, 0},{-h, i, 0, 0},{0, 0, 1, 0},{0, 0, 0, 1}};
{h, i, 0, 1}.T
```

Code in section on Top Vector.

```
(* display two cubes as a stereo pair *)
Clear[Trg, Tlf, pt, e, r, qt];
Tlf={{1,0,0,0},{0,1,0,0},{0,0,0,r},{e,0,0,1}};
Trg={{1,0,0,0},{0,1,0,0},{0,0,0,r},{-e,0,0,1}};
pt={{1,1,1,1},{-1,1,1,1},{1,-1,1,1},{-1,-1,1,1},
 {1,1,-1,1},{-1,1,-1,1},{1,-1,-1,1},
 {-1,-1,-1,1},{1,1,1,1}}; e=.1; r=3;
qt=Table[0, {i,9},{j,4}];
Do[qt[[i]]=pt[[i]].Tlf, {i,1,9}]; (* use Tlf for other image *)
Do[qt[[i,1]]=qt[[i,1]]/qt[[i,4]], {i,1,9}];
Do[qt[[i,2]]=qt[[i,2]]/qt[[i,4]], {i,1,9}];
ListPlot[Table[{qt[[i,1]], qt[[i,2]]},{i,1,9}],
```

```
 PlotJoined->True, Axes->False]
```

Code for Figure 6.56.

Chapter 7

```
(* exercise for hemispherical fisheye projection *)
k=1;
scal[q_]:=(k Tan[ArcTan[q/k]/2])/q;
{scal[1.],scal[10.], scal[100.], scal[1000.], scal[10000.]}
```

(In Exercise). Code to compute the scale factors for several $|\mathbf{P}|$ values from 1 to 10,000.

```
(* hemispherical fisheye projection *)
Clear[k, n, P, Q, L]
k=10; n=50;
scal[q_]:=(k Tan[ArcTan[q/k]/2])/q;
P=Table[{Random[Real,{-10.,10.}], Random[Real,{-10., 10.}]},{n}];
Q=Table[Sqrt[P[[i]].P[[i]]], {i, n}];
L=Table[Line[{P[[i]], scal[Q[[i]]] P[[i]]}], {i, n}];
Show[Graphics[L], Graphics[Circle[{0, 0}, 10]],
 Graphics[Point[{0, 0}]],  AspectRatio -> 1]
```

Figure 7.3. Moving Points in Hemispherical Fisheye Projection.

```
k = 10;
angl = {22.5, 45., 67.5, 89.};
k Tan[angl Degree]
k Tan[angl/2 Degree]
```

Table 7.6.

```
k = 10; n = 50; scal[q_] := (k Tan[ArcTan[q/k]/2])/q;
P = Table[{Random[Real, {-10.,10.}], Random[Real, {-10.,10.}]}, {n}];
x = -5; y = 5; (* Location of viewer *)
Pt = P - Table[{x, y}, {n}];
Q = Table[Sqrt[Pt[[i]].Pt[[i]]], {i, n}];
L = Table[Line[{P[[i]]+{x, y}, (scal[Q[[i]]] P[[i]])+{x, y}}], {i, n}];
Show[Graphics[L], Graphics[Circle[{0, 0}, k]],
 Graphics[{AbsolutePointSize[5], Point[{0, 0}]}],
 Graphics[{AbsolutePointSize[5], Point[{x, y}]}],
 AspectRatio -> Automatic, PlotRange -> All]
```

Figure 7.12. Off-Axis Fisheye Projection and Code.

```
k=10.;
Table[k z/(z+k), {z,0,100,5}]
Table[%[[i+1]]-%[[i]], {i,1,20}]
Table[Point[{%%[[i]],0}], {i,1,21}];
Show[Graphics[%]]
```

Code in Section 7.12.

```
l=20.; r=0.1;
Table[l(1-(z r/(z+l))), {z,0,100,5}]
Table[%[[i]]-%[[i+1]], {i,1,20}]
Table[Line[{{i, 17}, {i, %%[[i]]}}], {i,1,21}]
```

```
Show[Graphics[%]]
```

Code to Compute the heights of the transformed telephone poles.

Chapter 8

```
(* non-barycentric weights example *)
Clear[p0,p1,g1,g2,g3,g4];
p0 = {0, 0}; p1 = {5, 6};
g1 = ParametricPlot[(1-t)^3 p0+t^3 p1, {t,0,1},
 PlotRange->All, DisplayFunction->Identity];
g3=Graphics[{Red, AbsolutePointSize[4], {Point[p0], Point[p1]}}];
p0 = {0, -1}; p1 = {5, 5};
g2=ParametricPlot[(1-t)^3 p0+t^3 p1, {t, 0, 1},
 PlotRange->All, PlotStyle->AbsoluteDashing[{2, 2}],
 DisplayFunction->Identity];
g4=Graphics[{Red, AbsolutePointSize[6], {Point[p0], Point[p1]}}];
Show[g2, g1, g3, g4, PlotRange->All, AspectRatio->.5]
```

Figure 8.9. Effect of Nonbarycentric Weights.

```
Clear[points];
points={{0,1},{1,1.1},{2,1.2},{3,3},{4,2.9},{5,2.8},{6,2.7}};
InterpolatingPolynomial[points,x];
Interpolation[points,InterpolationOrder->3];
Show[ListPlot[points,Prolog->AbsolutePointSize[5]],
 Plot[%%,{x,0,6},PlotStyle->Dashing[{0.05,0.05}]],
 Plot[%[x],{x,0,6}]]
```

Figure 8.12. Polynomial and Spline Fit.

```
Compute P(0), dP, ddP, and dddP;
P = P(0);
for t:=0 to 1 step Δt do
PN:=P+dP; dP:=dP+ddP; ddP:=ddP+dddP;
line(P,PN);
P:=PN;
endfor;
```

Code in Section 8.8.1.

```
for u:=0 to 1 step 0.2 do
 begin
 for w:=0 to 1 step 0.01 do
  begin
  SurfacePoint(u,w,x,y,z);
  PersProj(x,y,z,xs,ys);
  Pixel(xs,ys,color)
  end;
 end;

 for w:=0 to 1 step 0.2 do
 begin
 for u:=0 to 1 step 0.01 do
  begin
  SurfacePoint(u,w,x,y,z);
  PersProj(x,y,z,xs,ys);
  Pixel(xs,ys,color)
  end;
```

```
end;
```

Figure 8.22. Procedure for a Wire-Frame Surface.

Chapter 9

```
(* a bilinear surface patch *)
Clear[bilinear,pnts,u,w];
<<:Graphics:ParametricPlot3D.m;
pnts=ReadList["Points",{Number,Number,Number}, RecordLists->True];
bilinear[u_,w_]:=pnts[[1,1]](1-u)(1-w)+pnts[[1,2]]u(1-w)\
 +pnts[[2,1]]w(1-u)+pnts[[2,2]]u w;
Simplify[bilinear[u,w]]
g1=Graphics3D[{AbsolutePointSize[5],Table[Point[pnts[[i,j]]],{i,1,2},{j,1,2}]}];
g2=ParametricPlot3D[bilinear[u,w],{u,0,1,.05},{w,0,1,.05}];
Show[g1,g2, PlotRange->All, ViewPoint->{0.063, -1.734, 2.905}];
{{0, 0, 1}, {1, 1, 1}, {1, 0, 0}, {0, 1, 0}}
{u + w - 2 u w, u, 1 - w}
```

Figure 9.7. A Bilinear Surface.

```
(* Another bilinear surface example *)
ParametricPlot3D[{0.5(1-u)w+u,w,(1-u)(1-w)}, {u,0,1},{w,0,1},
ViewPoint->{-0.846, -1.464, 3.997}];
```

Figure 9.8. A Bilinear Surface.

```
(* A Triangular bilinear surface example *)
ParametricPlot3D[{u(1-w),w,(1-u)(1-w)}, {u,0,1},{w,0,1},
ViewPoint->{-2.673, -3.418, 0.046}];
```

Figure 9.9. A Triangular Bilinear Surface.

```
(*A lofted surface example.Bottom boundary curve is straight*)
pnts={{-1,-1,0},{1,-1,0},{-1,1,0},{0,1,1},{1,1,0}};
g1=Graphics3D[{AbsolutePointSize[5],Table[Point[pnts[[i]]],{i,1,5}]}];
g2=ParametricPlot3D[{2u-1,2w-1,4u w (1-u)},{u,0,1},{w,0,1},
 AspectRatio->Automatic,Ticks->{{0,1},{0,1},{0,1}}];
Show[g1,g2,ViewPoint->{-0.139,-1.179,1.475}]
```

Figure Ans.32. A Lofted Surface.

```
Clear[loftedSurf]; (* double helix as a lofted surface *)
loftedSurf:={Cos[u],Sin[u],u}(1-w)+{Cos[u+Pi],Sin[u+Pi],u}w;
ParametricPlot3D[loftedSurf, {u,0,Pi,.1},{w,0,1},
Ticks->False, ViewPoint->{-2.640, -0.129, 0.007}]
```

Figure 9.12. The Double Helix as a Lofted Surface.

```
(* Another lofted surface example *)
<<:Graphics:ParametricPlot3D.m
Clear[ls];
ls=Simplify[{8u^3-12u^2+6u-1,4u^3-9u^2+6u,0}(1-w)+{2u-1,4u(u-1),1}w];
ParametricPlot3D[ls, {u,0,1,.1},{w,0,1,.1}, Compiled->False,
 ViewPoint->{-0.139, -1.179, 1.475}, DefaultFont->{"cmr10", 10},
 AspectRatio->Automatic, Ticks->{{0,1},{0,1},{0,1}}];
```

Figure 9.13. A Lofted Surface Patch.

Chapter 10

```
Solve[{d==p1,
a al^3+b al^2+c al+d==p2,
```

```
a be^3+b be^2+c be+d==p3,
a+b+c+d==p4},{a,b,c,d}];
ExpandAll[Simplify[%]]
```

Code to Solve the Generalized Form of Equation (10.6).

```
(* 3-point Lagrange polynomial (uniform and nonunif) *)
Clear[T,H,B,d0,d1];
d0=1; d1=1;
T={t^2,t,1};
H={{1/(d0(d0+d1)),-1/(d0 d1),1/(d1(d0+d1))},
{-1/(d0+d1)-1/d0,1/d0+1/d1,-1/d1+1/(d0+d1)},{1,0,0}};
B={{1,0},{1.3,.5},{4,0}};
Simplify[T.H.B];
C1=ParametricPlot[T.H.B,{t,0,d0+d1},
 PlotStyle->AbsoluteDashing[{2,2}],DisplayFunction->Identity];
d0=.583; d1=2.75;
H={{1/(d0(d0+d1)),-1/(d0 d1),1/(d1(d0+d1))},
{-1/(d0+d1)-1/d0,1/d0+1/d1,-1/d1+1/(d0+d1)},{1,0,0}};
Simplify[T.H.B];
C2=ParametricPlot[T.H.B,{t,0,d0+d1}];
Show[C1, C2, PlotRange->All]
```

Figure 10.1. Three-Point Lagrange Polynomials.

```
 (* 3-point Lagrange polynomial (3 examples of nonuniform) *)
Clear[T,H,B,d0,d1,C1,C2,C3];
d0=1.414; d1=1.415; (* d1=0.5|P2-P1| *)
T={t^2,t,1};
H={{1/(d0(d0+d1)),-1/(d0 d1),1/(d1(d0+d1))},
{-1/(d0+d1)-1/d0,1/d0+1/d1,-1/d1+1/(d0+d1)},{1,0,0}};
B={{1,1},{2,2},{4,0}};
Simplify[T.H.B]
C1=ParametricPlot[T.H.B,{t,0,d0+d1}];
d1=2.83; (* d1=|P2-P1| *)
H={{1/(d0(d0+d1)),-1/(d0 d1),1/(d1(d0+d1))},
{-1/(d0+d1)-1/d0,1/d0+1/d1,-1/d1+1/(d0+d1)},{1,0,0}};
Simplify[T.H.B]
C2=ParametricPlot[T.H.B,{t,0,d0+d1}];
d1=5.66;  (* d1=2|P2-P1| *)
H={{1/(d0(d0+d1)),-1/(d0 d1),1/(d1(d0+d1))},
{-1/(d0+d1)-1/d0,1/d0+1/d1,-1/d1+1/(d0+d1)},{1,0,0}};
Simplify[T.H.B]
C3=ParametricPlot[T.H.B,{t,0,d0+d1}];
Show[C1,C2,C3, PlotRange->All]
(* (1/24,-1/8)t^3+(-1/3,3/4)t^2+(1,-1)t *)
```

Figure 10.2. Three-Point Nonuniform Lagrange Polynomials.

```
(* Plot quadratic and cubic Lagrange basis functions *)
lagq={t^2,t,1}.{{1/2,-1,1/2}, {-3/2,2,-1/2}, {1,0,0}};
Plot[{lagq[[1]],lagq[[2]],lagq[[3]]}, {t,0,2}, PlotRange->All,
 AspectRatio->1]
lagc={t^3,t^2,t,1}.{{-9/2,27/2,-27/2,9/2},
 {9,-45/2,18,-9/2}, {-11/2,9,-9/2,1}, {1,0,0,0}};
Plot[{lagc[[1]], lagc[[2]], lagc[[3]], lagc[[4]]}, {t,0,1},
 PlotRange -> All, AspectRatio -> 1]
```

Figure 10.3. (a) Quadratic and (b) Cubic Lagrange Basis Functions.

```
(* Biquadratic patch for 9 points *)
Clear[T,pnt,M,g1,g2];
T[t_]:={t^2,t,1};
pnt={{{0,0,0},{1,0,0},{2,0,0}}, {{0,1,0},{1,1,1},{2,1,-.5}},
 {{0,2,0},{1,2,0},{2,2,0}}};
M={{2,-4,2},{-3,4,-1},{1,0,0}};
```

```
g2=Graphics3D[{Red,AbsolutePointSize[6],
Table[Point[pnt[[i,j]]],{i,1,3},{j,1,3}] }];
comb[i_]:=(T[u].M.pnt)[[i]](Transpose[M].T[w])[[i]];
g1=ParametricPlot3D[comb[1]+comb[2]+comb[3], {u,0,1},{w,0,1}];
Show[g1,g2, ViewPoint->{1.391, -2.776, 0.304}, PlotRange->All]
```

Figure 10.4. A Biquadratic Surface Patch Example.

```
(* BiCubic patch for 16 points *)
Clear[T,pnt,M,g1,g2];
T[t_]:={t^3,t^2,t,1};
pnt = {{{0,0,0},{1,0,0},{2,0,0},{3,0,0}},
{{0,1,0},{1,1,1}, {2,1,-.5},{3,1,0}},
{{0,2,-.5},{1,2,0},{2,2,.5},{3,2,0}},
{{0,3,0},{1,3,0},{2,3,0},{3,3,0}}};
M={{-4.5,13.5,-13.5,4.5},{9,-22.5,18,-4.5},{-5.5,9,-4.5,1},{1,0,0,0}};
g2=Graphics3D[{Red, AbsolutePointSize[6],
 Table[Point[pnt[[i,j]]], {i,1,4}, {j,1,4}]}];
comb[i_]:=(T[u].M.pnt)[[i]] (Transpose[M].T[w])[[i]];
g1=ParametricPlot3D[comb[1]+comb[2]+comb[3]+comb[4],{u,0,1},{w,0,1}];
Show[g1, g2, PlotRange->All]
```

Figure 10.6. A Bicubic Surface Patch Example.

```
Clear[Nh,p,pnts,U,W];
p00={0,0,0}; p10={1,0,1}; p20={2,0,1}; p30={3,0,0};
p01={0,1,1}; p11={1,1,2}; p21={2,1,2}; p31={3,1,1};
p02={0,2,1}; p12={1,2,2}; p22={2,2,2}; p32={3,2,1};
p03={0,3,0}; p13={1,3,1}; p23={2,3,1}; p33={3,3,0};
Nh={{-4.5,13.5,-13.5,4.5},{9,-22.5,18,-4.5},
 {-5.5,9,-4.5,1},{1,0,0,0}};
pnts={{p33,p32,p31,p30},{p23,p22,p21,p20},
 {p13,p12,p11,p10},{p03,p02,p01,p00}};
U[u_]:={u^3,u^2,u,1};   W[w_]:={w^3,w^2,w,1};
(* prt [i] extracts component i from the 3rd dimen of P *)
prt[i_]:=pnts[[Range[1,4],Range[1,4],i]];
p[u_,w_]:={U[u].Nh.prt[1].Transpose[Nh].W[w],
 U[u].Nh.prt[2].Transpose[Nh].W[w], \
 U[u].Nh.prt[3].Transpose[Nh].W[w]};
 g1=ParametricPlot3D[p[u,w], {u,0,1},{w,0,1},
 Compiled->False, DisplayFunction->Identity];
g2=Graphics3D[{AbsolutePointSize[2],
 Table[Point[pnts[[i,j]]],{i,1,4},{j,1,4}]}];
Show[g1,g2, ViewPoint->{-2.576, -1.365, 1.718}]
```

Figure Ans.33. An Interpolating Bicubic Surface Patch and Code.

```
Clear[p0,p1,p2,p3,basis,fourP,g0,g1,g2,g3,g4,g5];
p0[u_]:={u,0,Sin[Pi u]};p1[u_]:={u,1+u/10,Sin[Pi (u+.1)]};
p2[u_]:={u,2,Sin[Pi (u+.2)]};p3[u_]:={u,3+u/10,Sin[Pi (u+.3)]};
(*matrix ‘basis’ has dimensions 4x4x3*)
basis:={{p0[0],p0[.33],p0[.67],p0[1]},{p1[0],p1[.33],p1[.67],p1[1]},
{p2[0],p2[.33],p2[.67],p2[1]},{p3[0],p3[.33],p3[.67],p3[1]}};
fourP:=(*basis matrix for a 4-point curve*){{-4.5,13.5,-13.5,4.5},
 {9,-22.5,18,-4.5},{-5.5,9,-4.5,1},{1,0,0,0}};
prt[i_]:=
(*extracts component i from the 3rd dimen of ‘basis‘*)
```

```
basis[[Range[1,4],Range[1,4],i]];
coord[i_]:=(*calc.the 3 parametric components of the surface*)
{u^3,u^2,u,1}.fourP.prt[i].Transpose[fourP].{w^3,w^2,w,1};
g0=ParametricPlot3D[p0[u],{u,0,1}];
g1=ParametricPlot3D[p1[u],{u,0,1}];
g2=ParametricPlot3D[p2[u],{u,0,1}];
g3=ParametricPlot3D[p3[u],{u,0,1}];
g4=Graphics3D[{Red, AbsolutePointSize[6],Table[Point[basis[[i,j]]],
{i,1,4},{j,1,4}]}];
g5=ParametricPlot3D[{coord[1],coord[2],coord[3]},{u,0,1},{w,0,1}];
Show[ g0,g1,g2,g3,ViewPoint->{-2.576,-1.365,1.718},
 Ticks->False, PlotRange -> All]
Show[g4,g5,ViewPoint->{-2.576,-1.365,1.718}]
```

Figure 10.7. A Four-Curve Surface.

```
<<:Graphics:ParametricPlot3D.m;
Clear[p00,p01,p10,p11,pu0,pu1,p0w,p1w];
p00:={0,0,0}; p01:={0,1,0};
p10:={1,0,0}; p11:={1,1,0};
pu0:={u,0,Sin[Pi u]};
pu1:={u,1,Sin[Pi u]};
p0w:={0,w,Sin[Pi w]};
p1w:={1,w,Sin[Pi w]};
Simplify[
{1-u,u}.{p0w,p1w}+{1-w,w}.{pu0,pu1}
-p00(1-u)(1-w)-p01(1-u)w
-p10(1-w)u-p11 u w]
ParametricPlot3D[%,
{u,0,1,.2},{w,0,1,.2},
PlotRange->All,
AspectRatio->Automatic,
RenderAll->False,
Ticks->{{1},{0,1},{0,1}},
Prolog->AbsoluteThickness[.4]]
```

Figure 10.8. A Coons Surface.

```
p00={-1,-1,0};p01={-1,1,0};p10={1,-1,0};p11={1,1,0};
pnts={p00,p01,p10,p11,{1,-1/2,1/2},{1,1/2,-1/2},
 {0,-1,-1/2},{0,1,1/2}};
p0w[w_]:={-1,2w-1,0};
p1w[w_]:={1,(-4-w+27w^2-18w^3)/4,27(w-3w^2+2w^3)/4};
pu0[u_]:={2u-1,-1,2u^2-2u};
pu1[u_]:={2u-1,1,-2u^2+2u};
p[u_,w_]:=(1-u)p0w[w]+u p1w[w]+(1-w)pu0[u]+w pu1[u]-
 p00(1-u)(1-w)-p01 (1-u)w-p10 u (1-w)-p11 u w;
g1=Graphics3D[{Red, AbsolutePointSize[6],
 Table[Point[pnts[[i]]],{i,1,8}]}];
g2=ParametricPlot3D[p[u,w],{u,0,1},{w,0,1},
 Ticks->{{-1,1},{-1,1},{-1,1}}];
Show[g1,g2]
```

Figure 10.10. A Coons Surface Patch and Code.

```
(*Triangular Coons patch*)
Clear[T,M,g1,g2];
T[t_]:={1+2t^3-3t^2,3t^2-2t^3,1};
p00={0,0,0};p10={2,0,0};p11={1,1,0};
M={{-p00,-p11,{w,w,4w (1-w)}},{-p10,-p11,{2-w,w,4w (1-w)}},
 {{2u,0,4u (u-1)},p11,{0,0,0}}};
g2=Graphics3D[{Red, AbsolutePointSize[6],
 Point[p00],Point[p10],Point[p11]}];
comb[i_]:=(T[u].M)[[i]] T[w][[i]];
g1=ParametricPlot3D[comb[1]+comb[2]+comb[3],{u,0,1},{w,0,1}];
Show[g1,g2]
```

Figure 10.14. A Triangular Coons Surface Patch Example.

```
b[u_,w_]:={0,1/2,1}(1-u)(1-w)+{1,1/2,1}(1-u)w
 +{0,3/2,1}(1-w)u+{1,3/2,1}u w;
H={{2,-2,1,1},{-3,3,-2,-1},{0,0,1,0},{1,0,0,0}};
lu0={u^3,u^2,u,1}.H.{{0,0,0},{0,1/2,1},{0,0,1},{0,1,0}};
lu1={u^3,u^2,u,1}.H.{{1,0,0},{1,1/2,1},{0,0,1},{0,1,0}};
l[u_,w_]:=lu0(1-w)+lu1 w;
fu0={u^3,u^2,u,1}.H.{{3/2,1/2,0},{1,1/2,1},{0,0,1},{-1,0,0}};
fu1={u^3,u^2,u,1}.H.{{3/2,3/2,0},{1,3/2,1},{0,0,1},{-1,0,0}};
f[u_,w_]:=fu0(1-w)+fu1 w;
cu0={u^3,u^2,u,1}.H.{{1,0,0},{3/2,1/2,0},{1,0,0},{0,1,0}};
cu1={1,1/2,1};
c0w={w^3,w^2,w,1}.H.{{1,0,0},{1,1/2,1},{0,0,1},{0,1,0}};
c1w={w^3,w^2,w,1}.H.{{3/2,1/2,0},{1,1/2,1},{0,0,1},{-1,0,0}};
c[u_,w_]:=(1-u)c0w+u c1w+(1-w)cu0+w cu1 \
 -(1-u)(1-w){1,0,0}-u(1-w){3/2,1/2,0}-w(1-u)cu1- u w cu1;
g1=ParametricPlot3D[b[u,w], {u,0,1},{w,0,1}]
g2=ParametricPlot3D[l[u,w], {u,0,1},{w,0,1}]
g3=ParametricPlot3D[f[u,w], {u,0,1},{w,0,1}]
g4=ParametricPlot3D[c[u,w], {u,0,1},{w,0,1}]
Show[g1,g2,g3,g4, PlotRange -> All]
```

Figure 10.15. Bilinear, Lofted, and Coons Surface Patches.

Chapter 11

```
Clear[T,H,B]; (* Hermite Interpolation *)
T={t^3,t^2,t,1};
H={{2,-2,1,1},{-3,3,-2,-1},{0,0,1,0},{1,0,0,0}};
B={{0,0},{2,1},{1,1},{1,0}};
ParametricPlot[T.H.B,{t,0,1},PlotRange->All]
```

Code to display a single Hermite curve segment.

```
Solve[{a (1/3)^3+b (1/3)^2+p1t (1/3)+d==p1,
a (2/3)^3+b (2/3)^2+p1t (1/3)+d==p2, 3a+2b+p1t==p2t}, {a,b,d}],
```

(In Exercise).

```
(* Hermite 3D example *)
Clear[T,H,B];
T={t^3,t^2,t,1};
H={{2,-2,1,1},{-3,3,-2,-1},{0,0,1,0},{1,0,0,0}};
B={{0,0,0},{1,1,1},{1,0,0},{0,1,0}};
ParametricPlot3D[T.H.B,{t,0,1},
```

```
 ViewPoint->{-0.846, -1.464, 3.997}];
(* ViewPoint->{3.119, -0.019, 0.054} alt view *)
```

Figure 11.4. A Hermite Curve Segment in Space.

```
Clear[T,H,B]; (* Nonuniform Hermite segments *)
T={t^3,t^2,t,1};
H={{2,-2,1,1},{-3,3,-2,-1},{0,0,1,0},{1,0,0,0}};
B[delta_]:={{0,0},{2,0},delta{2,1},delta{2,-1}};
g1=ParametricPlot[T.H.B[0.5],{t,0,1}];
g2=ParametricPlot[T.H.B[1],{t,0,1}];
g3=ParametricPlot[T.H.B[1.5],{t,0,1}];
Show[g1,g2,g3, PlotRange->All]
```

Figure 11.5. Three Nonuniform Hermite Segments.

```
(*Two Ferguson patches*)
F1[t_]:=2t^3-3t^2+1;F2[t_]:=-2t^3+3t^2;
F3[t_]:=t^3-2t^2+t;F4[t_]:=t^3-t^2;
F[t_]:={F1[t],F2[t],F3[t],F4[t]};
p00={0,0,0};p01={0,1,0};pu00={1,0,1};
pw00={0,1,1};pu01={1,0,1};pw01={0,1,0};
p10={1,0,0};p11={1,1,0};pu10={1,0,-1};
pw10={0,1,0};pu11={1,0,-1};pw11={0,1,-1};
p20={2,0,0};p21={2,1,0};pu20={1,0,0};
pw20={0,1,0};pu21={1,0,0};pw21={0,1,0};
H={{p00,p01,pw00,pw01},{p10,p11,pw10,pw11},
{pu00,pu01,{0,0,0},{0,0,0}},{pu10,pu11,{0,0,0},{0,0,0}}};
prt[i_]:=H[[Range[1,4],Range[1,4],i]];
g1=
 ParametricPlot3D[{F[u].prt[1].F[w],F[u].prt[2].F[w],F[u].prt[3].F[w]},
 {u,0,.98},{w,0,1}];

H={{p10,p11,pw10,pw11},{p20,p21,pw20,pw21},{pu10,pu11,{0,0,0},{0,0,0}},
 {pu20,pu21,{0,0,0},{0,0,0}}};
g2=
 ParametricPlot3D[{F[u].prt[1].F[w],F[u].prt[2].F[w],F[u].prt[3].F[w]},
 {u,0.05,1},{w,0,1}];

g3=Graphics3D[{Red, AbsolutePointSize[6],
 Point[p00],Point[p01],Point[p10],Point[p11],Point[p20],Point[p21]}];

Show[g1,g2,g3, PlotRange->All, ViewPoint->{0.322,1.342,0.506}]
```

Figure 11.15. Two Ferguson Surface Patches.

Chapter 12

```
(* tilted helix as a periodic curve *)
ParametricPlot3D[{.05t+Cos[t],Sin[t],.1t},{t,0,10Pi},
 Ticks->{{-1,0,1,2},{-1,0,1},{0,1,2,3}},
 PlotPoints->100,PlotStyle->Red]
```

Figure 12.3. A Tilted Helix as a Periodic Curve.

```
(* Nonuniform cubic spline example *)
C1:=ParametricPlot[{1/24,-1/8}t^3+{-1/3,3/4}t^2+{1,-1}t, {t,0,2}];
C2:=ParametricPlot[{-1/12,0}t^2+{1/6,1/2}t+{1,0}, {t,0,2}];
C3:=ParametricPlot[-{1/24,1/8}t^3+{-1/12,0}t^2+{-1/6,1/2}t+{1,1}, {t,0,2}];
Show[C1, C2, C3, PlotRange->All, AspectRatio->Automatic]
```

Figure 12.5. A Nonuniform Cubic Spline Example.

```
(*quadratic spline example*)
C1:=ParametricPlot[{t,t^2-t},{t,0,1}];
C2:=ParametricPlot[{-t^2+t+1,t},{t,0,1}];
C3:=ParametricPlot[{-t+1,-t^2+t+1},{t,0,1}];
C4=Graphics[{Red, AbsolutePointSize[6],Point[{0,0}],
```

```
 Point[{1,0}],Point[{1,1}],Point[{0,1}]}];
Show[C1,C2,C3,C4,PlotRange->All, AspectRatio->Automatic]
```

Figure 12.11. A Quadratic Spline Example.

```
(* Cardinal spline example *)
T={t^3,t^2,t,1};
H[s_]:={{-s,2-s,s-2,s},{2s,s-3,3-2s,-s},{-s,0,s,0},{0,1,0,0}};
B={{1,3},{2,0},{3,2},{2,3}};
s=3/6; (* T=0 *)
g1=ParametricPlot[T.H[s].B,{t,0,1}];
s=2/6; (* T=1/3 *)
g2=ParametricPlot[T.H[s].B,{t,0,1}];
s=1/6; (* T=2/3 *)
g3=ParametricPlot[T.H[s].B,{t,0,1}];
s=0; (* T=1 *)
g4=ParametricPlot[T.H[s].B,{t,0,1}];
g5=Graphics[{AbsolutePointSize[4], Table[Point[B[[i]]],{i,1,4}] }];
Show[g1,g2,g3,g4,g5, PlotRange->All]
```

Figure 12.13. A Cardinal Spline Example.

```
0 0 0   1   0 0   2     0 0   3 0 0
0 1 0  .5  .5 1   2.5  .5 0   3 1 0
0 2 0  .5 2.5 0   2.5 2.5 1   3 2 0
0 3 0   1   3 0   2     3 0   3 3 0
```

Coordinates for 16 points in file CRpoints.

```
0 0 0  1  0 0 2   0 0 3 0 0
0 1 0 .5 .5 1 2.5 .5 0 3 1 0
0 2 0 .5 2.5 0 2.5 2.5 1 3 2 0
0 3 0  1 3  0 2  3  0 3 3 0

Clear[Pt,Bm,CRpatch,g1,g2];
Pt=ReadList["CRpoints",{Number,Number,Number},RecordLists->True];
Bm:={{-.5,1.5,-1.5,.5},{1,-2.5,2,-.5},{-.5,0,.5,0},{0,1,0,0}};
CRpatch[i_]:=(*1st patch,rows 1-4*)
 {u^3,u^2,u,1}.Bm.Pt[[{1,2,3,4},{1,2,3,4},i]].
 Transpose[Bm].{w^3,w^2,w,1};
g1=Graphics3D[{Red, AbsolutePointSize[6],
 Table[Point[Pt[[i,j]]],{i,1,4},{j,1,4}]}];
g2=ParametricPlot3D[{CRpatch[1],CRpatch[2],CRpatch[3]},
 {u,0,.98},{w,0,1}];
Show[g1,g2,ViewPoint->{-4.322,0.242,0.306},PlotRange->All]
```

Figure 12.17. A Catmull–Rom Surface Patch.

```
0 0 0  1  0 0 2   0 0 3 0 0
0 1 0 .5 .5 1 2.5 .5 0 3 1 0
0 2 0 .5 2.5 0 2.5 2.5 1 3 2 0
0 3 0  1 3  0 2  3  0 3 3 0
0 4 0  1 4  0 2  4  0 3 4 0

Clear[Pt,Bm,CRpatch,CRpatchM,g1,g2,g3];
Pt=ReadList["CRpoints",{Number,Number,Number},RecordLists->True];
Bm:={{-.5,1.5,-1.5,.5},{1,-2.5,2,-.5},{-.5,0,.5,0},{0,1,0,0}};
CRpatch[i_]:=(*1st patch,rows 1-4*){u^3,u^2,u,1}.Bm.
 Pt[[{1,2,3,4},{1,2,3,4},i]].Transpose[Bm].{w^3,w^2,w,1};
CRpatchM[i_]:=(*2nd patch,rows 2-5*){u^3,u^2,u,1}.Bm.
 Pt[[{2,3,4,5},{1,2,3,4},i]].Transpose[Bm].{w^3,w^2,w,1};
g1=Graphics3D[{Red,AbsolutePointSize[6],
 Table[Point[Pt[[i,j]]],{i,1,5},{j,1,4}]}];
g2=ParametricPlot3D[{CRpatch[1],CRpatch[2],CRpatch[3]},
 {u,0,.98},{w,0,1}];
g3=ParametricPlot3D[{CRpatchM[1],CRpatchM[2],CRpatchM[3]},
 {u,0,1},{w,0,1}];
```

```
Show[g1,g2,g3,PlotRange->All]
```

Figure 12.18. Two Catmull–Rom Surface Patches.

```
(* A Catmull-Rom surface with tension *)
Clear[Pt,Bm,CRpatch,g1,g2,s];
Pt={{{0,3,0},{1,3,0},{2,3,0},{3,3,0}},
 {{0,2,0},{.1,2,.9},{2.9,2,.9},{3,2,0}},
 {{0,1,0},{.1,1,.9},{2.9,1,.9},{3,1,0}},
 {{0,0,0},{1,0,0},{2,0,0},{3,0,0}}};
Bm:={{-s,2-s,s-2,s},{2s,s-3,3-2s,-s},{-s,0,s,0},{0,1,0,0}};
CRpatch[i_]:=(*rows 1-4*){u^3,u^2,u,1}.Bm.
 Pt[[{1,2,3,4},{1,2,3,4},i]].Transpose[Bm].{w^3,w^2,w,1};
g1=Graphics3D[{Red,AbsolutePointSize[6],
 Table[Point[Pt[[i,j]]],{i,1,4},{j,1,4}]}];
s=.4;
g2=ParametricPlot3D[{CRpatch[1],CRpatch[2],CRpatch[3]},
 {u,0,1},{w,0,1}];
Show[g1,g2,ViewPoint->{1.431,-4.097,0.011},PlotRange->All]
```

Figure 12.19. A Catmull–Rom Surface Patch With Tension.

```
Clear[T, H, B, pts, Pa, Pd, te, bi, co];
(*Kochanek Bartels 3+2 points*)
T = {t^3, t^2, t, 1};
H = {{2, -2, 1, 1}, {-3, 3, -2, -1}, {0, 0, 1, 0}, {1, 0, 0, 0}};
Pd[k_] := (1 - te[[k + 1]]) (1 + bi[[k + 1]]) (1 +
    co[[k + 1]]) (pts[[k + 1]] - pts[[k]])/
   2 + (1 - te[[k + 1]]) (1 - bi[[k + 1]]) (1 -
    co[[k + 1]]) (pts[[k + 2]] - pts[[k + 1]])/2;
Pa[k_] := (1 - te[[k + 2]]) (1 + bi[[k + 2]]) (1 -
    co[[k + 2]]) (pts[[k + 2]] - pts[[k + 1]])/
   2 + (1 - te[[k + 2]]) (1 - bi[[k + 2]]) (1 +
    co[[k + 2]]) (pts[[k + 3]] - pts[[k + 2]])/2;
pts := {{-1, -1}, {0, 0}, {4, 6}, {10, -1}, {11, -2}};
te = {0, 0, 0, 0, 0}; bi = {0, 0, 0, 0, 0}; co = {0, 0, 0, 0, 0};
B = {pts[[2]], pts[[3]], Pd[1], Pa[1]};
Simplify[T.H.B];
Simplify[D[T.H.B, t]];
g1 = ParametricPlot[T.H.B, {t, 0, 1}, PlotRange -> All];

B = {pts[[3]], pts[[4]], Pd[2], Pa[2]};
Simplify[T.H.B];
Simplify[D[T.H.B, t]];
g2 = ParametricPlot[T.H.B, {t, 0, 1}, PlotRange -> All];
g3 = Graphics[{Red, AbsolutePointSize[6],
  Table[Point[pts[[i]]], {i, 1, 5}]}];
Show[g1, g2, g3, PlotRange -> All]
```

Figure 12.24. Effects of the Three Parameters in the Kochanek–Bartels Spline.

```
pnts={{2,5},{2,8},{5,11},{8,8},{11,4},{14,8},{13,8},{11,10}};
t=Table[N[Sqrt[(pnts[[i+1,1]]-pnts[[i,1]])^2
 +(pnts[[i+1,2]]-pnts[[i,2]])^2],4],{i,1,7}]
Do[t[[i+1]]=t[[i+1]]+t[[i]], {i,1,6}];
t
t=t/t[[7]]
```

Figure Ans.37. Eight Experimental Points and Their Polygon.

```
t={-0.1,0.1,0.2,0.3,0.4,0.6,0.8,1.2};
al=0.1; be=0.2; n=8;
scale[t_]:=t+al (n-i)/(n-1)-be (i-1)/(n-1);
t=Table[scale[t[[i]]], {i,1,8}]
```

(In Exercise). Code to Produce the eight scaled values

$$0,\ 0.157143,\ 0.214286,\ 0.271429,\ 0.328571,\ 0.485714,\ 0.642857,\ 1.$$

Chapter 13

```
(* Just the base functions bern. Note how "pwr" handles 0^0 *)
Clear[pwr,bern];
pwr[x_,y_]:=If[x==0 && y==0, 1, x^y];
bern[n_,i_,t_]:=Binomial[n,i]pwr[t,i]pwr[1-t,n-i] (* t^i x (1-t)^(n-i) *)
Plot[Evaluate[Table[bern[5,i,t], {i,0,5}]], {t,0,1}];
```

Figure 13.2. The Bernstein Polynomials for $n = 2, 3, 4$.

```
(*Just the base functions bern.Note how "pwr" handles 0^0*)
Clear[pwr,bern,n,i,t]
pwr[x_,y_]:=If[x==0&&y==0,1,x^y];
bern[n_,i_,t_]:=Binomial[n,i]pwr[t,i]pwr[1-t,n-i]
(*t^i*(1-t)^(n-i)*)
Plot[Evaluate[Table[bern[5,i,t],{i,0,5}]],{t,0,1}]
Clear[i,t,pnts,pwr,bern,bzCurve,g1,g2];
(*Cubic Bezier curve
 either read points from file
 pnts=ReadList["DataPoints",{Number,Number}];*)
 or enter them explicitly*)
pnts={{0,0},{.7,1},{.3,1},{1,0}};
(*4 points for a cubic curve*)
pwr[x_,y_]:=If[x==0&&y==0,1,x^y];
bern[n_,i_,t_]:=Binomial[n,i]pwr[t,i]pwr[1-t,n-i]
bzCurve[t_]:=Sum[pnts[[i+1]]bern[3,i,t],{i,0,3}]
g1=Graphics[{Red, AbsolutePointSize[6],
 Table[Point[pnts[[i]]],{i,1,4}]}];
g2=ParametricPlot[bzCurve[t],{t,0,1}];
Show[g1,g2,PlotRange->All]
```

Code to Plot the Bernstein Polynomials.

```
Clear[pnts,pwr,bern,bzCurve,g1,g2,g3];
(*General 3D Bezier curve*)
pnts={{1,0,0},{0,-3,0.5},{-3,0,0.75},{0,3,1},
 {3,0,1.5},{0,-3,1.75},{-1,0,2}};
n=Length[pnts]-1;
pwr[x_,y_]:=If[x==0&&y==0,1,x^y];
bern[n_,i_,t_]:=Binomial[n,i]pwr[t,i]pwr[1-t,n-i]
 (*t^i x (1-t)^(n-i)*)
bzCurve[t_]:=Sum[pnts[[i+1]]bern[n,i,t],{i,0,n}];
g1=ParametricPlot3D[bzCurve[t],{t,0,1},DisplayFunction->Identity];
g2=Graphics3D[{AbsolutePointSize[2],Map[Point,pnts]}];
g3=Graphics3D[{AbsoluteThickness[2],
(*control polygon*)
 Table[Line[{pnts[[j]],pnts[[j+1]]}],{j,1,n}]}];
g4=Graphics3D[{AbsoluteThickness[1.5],
(*the coordinate axes*)
 Line[{{0,0,3},{0,0,0},{3,0,0},{0,0,0},{0,3,0}}]}];
Show[g1,g2,g3,g4,AspectRatio->Automatic,PlotRange->All,Boxed->False]
```

Plot a General 3D Bezier curve.

```
(*Heart-shaped Bezier curve*)n=9;ppr=130;
pnts={{0,0},{-ppr,70},{-ppr,200},{0,200},{250,0},{-250,0},
 {0,200},{ppr,200},{ppr,70},{0,0}};
pwr[x_,y_]:=If[x==0&&y==0,1,x^y];
bern[n_,i_,t_]:=Binomial[n,i]pwr[t,i]pwr[1-t,n-i]
```

```
bzCurve[t_]:=Sum[pnts[[i+1]]bern[n,i,t],{i,0,n}]
g1=ListPlot[pnts,PlotStyle->{Red,AbsolutePointSize[6]}];
g2=ParametricPlot[bzCurve[t],{t,0,1}];
g3=Graphics[{AbsoluteDashing[{1,2,5,2}],Line[pnts]}];
Show[g1,g2,g3,PlotRange->All]
```

Figure Ans.38. A Heart-Shaped Bézier Curve.

precalculate certain quantities;
$\mathbf{B}=\mathbf{P}_0$;
<u>for</u> t:=0 <u>to</u> 1 <u>step</u> Δt <u>do</u>
PlotPixel(B);
B:=B+dB; dB:=dB+ddB; ddB:=ddB+dddB;
<u>endfor</u>;

Code to Compute Forward Differences

Q1:=$3\Delta t$;
Q2:=Q1$\times\Delta t$; // $3\Delta^2 t$
Q3:=$\Delta^3 t$;
Q4:=2Q2; // $6\Delta^2 t$
Q5:=6Q3; // $6\Delta^3 t$
Q6:=$\mathbf{P}_0-2\mathbf{P}_1+\mathbf{P}_2$;
Q7:=$3(\mathbf{P}_1-\mathbf{P}_2)-\mathbf{P}_0+\mathbf{P}_3$;
B:=$\mathbf{P}_0$;
dB:=$(\mathbf{P}_1-\mathbf{P}_0)$Q1+Q6$\times$Q2+Q7$\times$Q3;
ddB:=Q6$\times$Q4+Q7$\times$Q5;
dddB:=Q7$\times$Q5;
<u>for</u> t:=0 <u>to</u> 1 <u>step</u> Δt <u>do</u>
Pixel(B);
B:=B+dB; dB:=dB+ddB; ddB:=ddB+dddB;
<u>endfor</u>;

```
n=3; Clear[q1,q2,q3,q4,q5,Q6,Q7,B,dB,ddB,dddB,p0,p1,p2,p3,tabl];
p0={0,1}; p1={5,.5}; p2={0,.5}; p3={0,1}; (* Four points *)
dt=.01; q1=3dt; q2=3dt^2; q3=dt^3; q4=2q2; q5=6q3;
Q6=p0-2p1+p2; Q7=3(p1-p2)-p0+p3;
B=p0; dB=(p1-p0) q1+Q6 q2+Q7 q3; (* space indicates *)
ddB=Q6 q4+Q7 q5; dddB=Q7 q5;     (* multiplication *)
tabl={};
Do[{tabl=Append[tabl,B], B=B+dB, dB=dB+ddB, ddB=ddB+dddB},
                                                     {t,0,1,dt}];
ListPlot[tabl];
```

Figure 13.4. A Fast Bézier Curve Algorithm.

```
(* New points for Bezier curve subdivision exercise *)
pnts={{0,1,1},{1,1,0},{4,2,0},{6,1,1}};
t=1/3;
pwr[x_,y_]:=If[x==0 && y==0, 1, x^y];
bern[n_,i_,t_]:=Binomial[n,i]pwr[t,i]pwr[1-t,n-i]
p01=Sum[pnts[[i+1]]bern[1,i,t], {i,0,1}]
p012=Sum[pnts[[i+1]]bern[2,i,t], {i,0,2}]
p0123=Sum[pnts[[i+1]]bern[3,i,t], {i,0,3}]
p0123=Sum[pnts[[3-3+i+1]]bern[3,i,t], {i,0,3}]
p123=Sum[pnts[[3-2+i+1]]bern[2,i,t], {i,0,2}]
p23=Sum[pnts[[3-1+i+1]]bern[1,i,t], {i,0,1}]
```

Figure Ans.41. Code to Compute Six New Points.

```
t=0;
TotSegLen=0; // total length of segments visited so far
L=0; // total length of polyline
for i=1 to k do L=L+|P_i - P_{i-1}|; endfor;
st=0; s=L/n; // size of a chunk
AddTable(0); // add initial value
for i=1 to k do // loop over k segments
 SegLen=|P_i - P_{i-1}|;
 TotSegLen=TotSegLen+SegLen;
 if(s-st≤SegLen)
 then // a chunk ends at this segment
  t=t+(s-st)/L;
  AddTable(t);
  while SegLen>s do // more chunks in
   t=t+s/L;         // this segment
   AddTable(t);
   SegLen=SegLen-s;
  endwhile;
  st=SegLen;
 else // entire segment is part of chunk
  st=st+SegLen;
 endif;
 t=t+TotSegLen/L;
endfor;
AddTable(1); // add final value
```

Figure 13.20. Measuring n Chunks on a Polyline.

```
(* Interpolating Bezier Curve: I *)
p0={1/2,0};p1={1/2,1/2};p2={0,1};
p3={1,3/2};p4={3/2,1};p5={1,1/2};
x1=p1+(p2-p0)/6; x2=p2+(p3-p1)/6;
x3=p3+(p4-p2)/6; y1=p2-(p3-p1)/6;
y2=p3-(p4-p2)/6; y3=p4-(p5-p3)/6;
c1[t_]:=Simplify[(1-t)^3 p1+3t (1-t)^2 x1+3t^2(1-t) y1+t^3 p2]
c2[t_]:=Simplify[(1-t)^3 p2+3t (1-t)^2 x2+3t^2(1-t) y2+t^3 p3]
c3[t_]:=Simplify[(1-t)^3 p3+3t (1-t)^2 x3+3t^2(1-t) y3+t^3 p4]
g1=ListPlot[{p0,p1,p2,p3,p4,p5,x1,x2,x3,y1,y2,y3},
 PlotStyle->{Red,AbsolutePointSize[6]}, AspectRatio->Automatic];
g2=ParametricPlot[c1[t],{t,0,.9}];
g3=ParametricPlot[c2[t],{t,0.1,.9}];
g4=ParametricPlot[c3[t],{t,0.1,1}];
Show[g1,g2,g3,g4,PlotRange->All]
```

Figure 13.24. An Interpolating Bézier Curve.

```
Clear[p0,p1,p2,p3,p4,p5,x0,x1,x2,x3,x4,y1,y2,y3,y4,y5,c1,
 c2,c3,c4,c5,g1,g2,g3,g4,g5,g6];
p0={1/2,0};p1={1/2,1/2};p2={0,1};p3={1,3/2};
p4={3/2,1};p5={1,1/2};
x0=p1-p0;y5=p4-p5;
x1=p1+(p2-p0)/2;x2=p2+(p3-p1)/2;
```

```
x3=p3+(p4-p2)/2;x4=p4+(p5-p3)/2;
y1=p1-(p2-p0)/2;y2=p2-(p3-p1)/2;
y3=p3-(p4-p2)/2;y4=p4-(p5-p3)/2;
c1[t_]:=Simplify[(1-t)^3 p0+3t (1-t)^2 x0+3t^2(1-t) y1+t^3 p1]
c2[t_]:=Simplify[(1-t)^3 p1+3t (1-t)^2 x1+3t^2(1-t) y2+t^3 p2]
c3[t_]:=Simplify[(1-t)^3 p2+3t (1-t)^2 x2+3t^2(1-t) y3+t^3 p3]
c4[t_]:=Simplify[(1-t)^3 p3+3t (1-t)^2 x3+3t^2(1-t) y4+t^3 p4]
c5[t_]:=Simplify[(1-t)^3 p4+3t (1-t)^2 x4+3t^2(1-t) y5+t^3 p5]
g1=ListPlot[{p0,p1,p2,p3,p4,p5,x0,x1,x2,x3,x4,y1,y2,y3,y4,y5},
 PlotStyle->{Red,AbsolutePointSize[6]},AspectRatio->Automatic];
g2=ParametricPlot[c1[t],{t,0,.95}];
g3=ParametricPlot[c2[t],{t,0.05,.95}];
g4=ParametricPlot[c3[t],{t,0.05,.95}];
g5=ParametricPlot[c4[t],{t,0.05,.95}];
g6=ParametricPlot[c5[t],{t,0.05,1}];
Show[g1,g2,g3,g4,g5,g6,PlotRange->All]
```

Figure 13.25. An Interpolating Bézier Curve: II.

```
q0={0,0}; q1={1,1}; q2={2,1}; q3={3,0};
p0=q0; p1={1,3/2}; p2={2,3/2}; p3=q3;
c[t_]:=(1-t)^3 p0+3t(1-t)^2 p1+3t^2(1-t) p2+t^3 p3
g1=ListPlot[{p0,p1,p2,p3,q1,q2},
 Prolog->AbsolutePointSize[4]];
g2=ParametricPlot[c[t], {t,0,1}];
Show[g1,g2, PlotRange->All]
```

Figure Ans.42. An Interpolating Bézier Curve: III.

```
(* Effects of varying weights in Rational Cubic Bezier curve *)
Clear[RatCurve,g1,g2,w];
pnts={{0,0},{.2,1},{.8,1},{1,0}};
w={1,1,1,1};(*Four weights for a cubic curve*)
pwr[x_,y_]:=If[x==0&&y==0,1,x^y];
bern[n_,i_,t_]:=Binomial[n,i]pwr[t,i]pwr[1-t,n-i] (*t^i*(1-t)^(n-i)*)
RatCurve[t_]:=Sum[(w[[i+1]]pnts[[i+1]]bern[3,i,t])/
 (Sum[w[[j+1]]bern[3,j,t],{j,0,3}]),{i,0,3}];
g1=ListPlot[pnts,PlotStyle->{Red,AbsolutePointSize[6]},
 AspectRatio->Automatic];
g2=ParametricPlot[RatCurve[t],{t,0,1},AspectRatio->Automatic];
w={1,2,1,1};(*change weights*)g3=ParametricPlot[RatCurve[t],
 {t,0,1},AspectRatio->Automatic];
w={1,3,1,1};(*increase w1*)g4=ParametricPlot[RatCurve[t],
 {t,0,1},AspectRatio->Automatic];
w={1,4,1,1};(*increase w1*)g5=ParametricPlot[RatCurve[t],
 {t,0,1},AspectRatio->Automatic];
Show[g1,g2,g3,g4,g5,PlotRange->All]

(* Effects of moving a control point in Rational Cubic Bezier curve *)
Clear[RatCurve,g1,g2,w];
pnts={{0,0},{.2,.8},{.8,.8},{1,0}};
w={1,1,1,1};(*Four weights for a cubic curve*)
 pwr[x_,y_]:=If[x==0&&y==0,1,x^y];
bern[n_,i_,t_]:=Binomial[n,i]pwr[t,i]pwr[1-t,n-i] (*t^i*(1-t)^(n-i)*)
RatCurve[t_]:=Sum[(w[[i+1]]pnts[[i+1]]bern[3,i,t])/
 (Sum[w[[j+1]]bern[3,j,t],{j,0,3}]),{i,0,3}];
g1=ListPlot[pnts,PlotStyle->{Red,AbsolutePointSize[6]},
 AspectRatio->Automatic];
g2=ParametricPlot[RatCurve[t],{t,0,1},AspectRatio->Automatic];
```

```
pnts={{0,0},{.2,.8},{.86,.86},{1,0}};
g3=ParametricPlot[RatCurve[t],{t,0,1},AspectRatio->Automatic];
pnts={{0,0},{.2,.8},{.93,.93},{1,0}};
g4=ParametricPlot[RatCurve[t],{t,0,1},AspectRatio->Automatic];
pnts={{0,0},{.2,.8},{1,1},{1,0}};
g5=ParametricPlot[RatCurve[t],{t,0,1},AspectRatio->Automatic];
Show[g1,g2,g3,g4,g5,PlotRange->All]
```

Figure Ans.43. Code for Figure 13.27.

```
Clear[BCtab,CBtab,Bern,den,b1,b2,t1,t2,c0,c1,c2,c3];
t1=0; t2=Pi/2; c0=2; c1=2.2; c2=1.6; c3=1;
den=Sin[t2-t1]; b1=Sin[t2-t]/den; b2=Sin[t-t1]/den;
Bern[t_]:=c0 b1^3+3 c1 b1^2 b2^1 Sin[t]+3 c2 b1^1 b2^2+c3 b2^3;
CBtab=Table[{Cos[t] Bern[t], Sin[t] Bern[t]}, {t,0,Pi/2,0.1}];
v={c0{Cos[0],Sin[0]}, c1{Cos[Pi/6],Sin[Pi/6]},
 c2{Cos[Pi/3],Sin[Pi/3]}, c3{Cos[Pi/2],Sin[Pi/2]}};
 v//N
c2=1.3;
BCtab=Table[{Cos[t] Bern[t], Sin[t] Bern[t]}, {t,0,Pi/2,0.1}];
Show[ListPlot[CBtab],ListPlot[BCtab], PlotRange->All,
 AspectRatio->Automatic]
```

(In Exercise). Calculate the four control points of the cubic circular curve defined by $\theta_1 = 0$, $\theta_2 = 90^\circ = \pi/2$, $c_0 = 2$, $c_1 = 1.2$, $c_2 = 1.6$, and $c_3 = 1$.

```
(* biquadratic bezier surface patch *)
Clear[pwr,bern,spnts,n,bzSurf,g1,g2];
n=2;
spnts={{{0,0,0},{1,0,1},{0,0,2}},{{1,1,0},{4,1,1},{1,1,2}},
 {{0,2,0},{1,2,1},{0,2,2}}};
(*Handle Indeterminate condition*)
pwr[x_,y_]:=If[x==0&&y==0,1,x^y];
bern[n_,i_,u_]:=Binomial[n,i]pwr[u,i]pwr[1-u,n-i]
bzSurf[u_,w_]:=Sum[bern[n,i,u] spnts[[i+1,j+1]] bern[n,j,w],
 {i,0,n},{j,0,n}]
g1=ParametricPlot3D[bzSurf[u,w],{u,0,1},{w,0,1},
 Ticks->{{0,1,4},{0,1,2},{0,1,2}}];
g2=Graphics3D[{Red,AbsolutePointSize[6],
 Table[Point[spnts[[i,j]]],{i,1,n+1},{j,1,n+1}]}];
Show[g1,g2,ViewPoint->{2.783,-3.090,1.243},PlotRange->All]
```

Figure 13.33. A Biquadratic Bézier Surface Patch.

```
(* A Bezier surface example. Given the six two-dimensional... *)
Clear[pnts,b1,b2,g1,g2,vlines,hlines];
pnts={{{0,1,0},{1,1,1},{2,1,0}},{{0,0,0},{1,0,0},{2,0,0}}};
b1[w_]:={1-w,w};b2[u_]:={(1-u)^2,2u (1-u),u^2};
comb[i_]:=(b1[w].pnts)[[i]] b2[u][[i]];
g1=ParametricPlot3D[comb[1]+comb[2]+comb[3],{u,0,1},{w,0,1},
 AspectRatio->Automatic,Ticks->{{0,1,2},{0,1},{0,.5}}];
g2=Graphics3D[{Red,AbsolutePointSize[6],
 Table[Point[pnts[[i,j]]],{i,1,2},{j,1,3}]}];
vlines=Graphics3D[{Green,AbsoluteThickness[2],
 Table[Line[{pnts[[1,j]],pnts[[2,j]]}],{j,1,3}]}];
hlines=Graphics3D[{Green,AbsoluteThickness[2],
 Table[Line[{pnts[[i,j]],pnts[[i,j+1]]}],{i,1,2},{j,1,2}]}];
Show[g1,g2,vlines,hlines,PlotRange->All]
```

Figure 13.34. A Lofted Bézier Surface Patch.

```
(* Degree elevation of a rect Bezier surface from 2x3 to 4x5 *)
Clear[p,q,r];
m=1; n=2;
```

```
p={{p00,p01,p02},{p10,p11,p12}}; (* array of points *)
r=Array[a, {m+3,n+3}]; (* extended array, still undefined *)
Part[r,1]=Table[a, {i,-1,m+2}];
Part[r,2]=Append[Prepend[Part[p,1],a],a];
Part[r,3]=Append[Prepend[Part[p,2],a],a];
Part[r,n+2]=Table[a, {i,-1,m+2}];
MatrixForm[r] (* display extended array *)
q[i_,j_]:=({i/(m+1),1-i/(m+1)}. (* dot product *)
 {{r[[i+1,j+1]],r[[i+1,j+2]]},{r[[i+2,j+1]],r[[i+2,j+2]]}}).
 {j/(n+1),1-j/(n+1)}
q[2,3] (* test *)

(* Degree elevation of a rect Bezier surface from 2x3 to 4x5 *)
Clear[p,r,comb];
m=1; n=2; (* set p to an array of 3D points *)
p={{{0,0,0},{1,0,1},{2,0,0}},{{0,1,0},{1,1,.5},{2,1,0}}};
r=Array[a, {m+3,n+3}]; (* extended array, still undefined *)
Part[r,1]=Table[{a,a,a}, {i,-1,m+2}];
Part[r,2]=Append[Prepend[Part[p,1],{a,a,a}],{a,a,a}];
Part[r,3]=Append[Prepend[Part[p,2],{a,a,a}],{a,a,a}];
Part[r,n+2]=Table[{a,a,a}, {i,-1,m+2}];
MatrixForm[r] (* display extended array *)
comb[i_,j_]:=({i/(m+1),1-i/(m+1)}.
{{r[[i+1,j+1]],r[[i+1,j+2]]},{r[[i+2,j+1]],r[[i+2,j+2]]}})[[1]]{j/(n+1),1-j/(n+1)}[[1]]+
({i/(m+1),1-i/(m+1)}.
{{r[[i+1,j+1]],r[[i+1,j+2]]},{r[[i+2,j+1]],r[[i+2,j+2]]}})[[2]]{j/(n+1),1-j/(n+1)}[[2]];
MatrixForm[Table[comb[i,j],{i,0,2},{j,0,3}]]
```

Figure 13.37. Code for Degree Elevation of a Rectangular Bézier Surface.

```
Clear[n,bern,p1,p2,g3,bzSurf,patch];
n=2;
p1={{{-2,2,2},{-2,2,0},{0,2,0}},{{-4,0,2},{-4,0,0},
 {0,0,0}},{{-2,-2,2},{-2,-2,0},{0,-2,0}}};
p2={{{0,2,0},{2,2,0},{2,2,-2}},{{0,0,0},{4,0,0},{4,0,-2}},
 {{0,-2,0},{2,-2,0},{2,-2,-2}}};
pwr[x_,y_]:=If[x==0&&y==0,1,x^y];
bern[n_,i_,u_]:=Binomial[n,i]pwr[u,i]pwr[1-u,n-i]
bzSurf[p_]:={Sum[p[[i+1,j+1,1]]bern[n,i,u]bern[n,j,w],
 {i,0,n,1},{j,0,n,1}],
 Sum[p[[i+1,j+1,2]]bern[n,i,u]bern[n,j,w],
  {i,0,n,1},{j,0,n,1}],
 Sum[p[[i+1,j+1,3]]bern[n,i,u]bern[n,j,w],
  {i,0,n,1},{j,0,n,1}]};
patch[s_]:=ParametricPlot3D[bzSurf[s],
 {u,0,1},{w,0.02,.98}];
g3=Graphics3D[{Red,AbsolutePointSize[6],
 Table[Point[p1[[i,j]]],{i,1,n+1},{j,1,n+1}]}];
g4=Graphics3D[{Red,AbsolutePointSize[6],
 Table[Point[p2[[i,j]]],{i,1,n+1},{j,1,n+1}]}];
Show[patch[p1],patch[p2],g3,g4,PlotRange->All]
```

Figure 13.39. Two Bézier Surface Patches.

```
(*Sphere made of 8 Bezier patches*)
Clear[u,w,patch];
al3=Sin[30. Degree];be3=Cos[30. Degree];
p00=p10=p20=p30={0,0,1};p03={1,0,0};p33={0,1,0};
t3={{be3,al3,0},{-al3,be3,0},{0,0,1}};
k=0.5523;
p13={1,k,0};p23={k,1,0};
```

```
p02={1,0,k};p01={k,0,1};
p32={0,1,k};p31={0,k,1};
p11=p01.t3;p12=p02.t3;
t6={{al3,be3,0},{-be3,al3,0},{0,0,1}};
p21=p01.t6;p22=p02.t6;
b30[t_]:=(1-t)^3;b31[t_]:=3t (1-t)^2;
b32[t_]:=3t^2(1-t);b33[t_]:=t^3;
patch[u_,w_]:=b30[w](b30[u]p00+b31[u]p01+b32[u]p02+
 b33[u]p03)+b31[w](b30[u]p10+b31[u]p11+b32[u]p12+
 b33[u]p13)+b32[w](b30[u]p20+b31[u]p21+b32[u]p22+
 b33[u]p23)+b33[w](b30[u]p30+b31[u]p31+b32[u]p32+b33[u]p33);
Factor[patch[u,w]]
ParametricPlot3D[patch[u,w],{u,0,1},{w,0,1},
 Prolog->AbsoluteThickness[.5],ViewPoint->{1.908,-3.886,0.306}]
```

Figure 13.41. Code for a Bézier Patch.

```
(* A Rational Bezier Surface *)
Clear[pwr,bern,spnts,n,m,wt,bzSurf,cpnts,patch,vlines,hlines,axes];
spnts={{{0,0,0},{1,0,1},{0,0,2}},{{1,1,0},{4,1,1},{1,1,2}},
 {{0,2,0},{1,2,1},{0,2,2}}};
m=Length[spnts[[1]]]-1;n=Length[Transpose[spnts][[1]]]-1;
wt=Table[1,{i,1,n+1},{j,1,m+1}];
wt[[2,2]]=5;
pwr[x_,y_]:=If[x==0&&y==0,1,x^y];
bern[n_,i_,u_]:=Binomial[n,i]pwr[u,i]pwr[1-u,n-i]
bzSurf[u_,w_]:=Sum[wt[[i+1,j+1]]spnts[[i+1,j+1]]bern[n,i,u]bern[m,j,w],
 {i,0,n},{j,0,m}]/Sum[wt[[i+1,j+1]]bern[n,i,u]bern[m,j,w],
 {i,0,n},{j,0,m}];
patch=ParametricPlot3D[bzSurf[u,w],{u,0,1},{w,0,1}];
cpnts=Graphics3D[{Red,AbsolutePointSize[6],
 (*control points*)
Table[Point[spnts[[i,j]]],{i,1,n+1},{j,1,m+1}]}];
vlines=Graphics3D[{Green,AbsoluteThickness[1],
(*control polygon*)
Table[Line[{spnts[[i,j]],spnts[[i+1,j]]}],{i,1,n},{j,1,m+1}]}];
hlines=Graphics3D[{Green,AbsoluteThickness[1],
 Table[Line[{spnts[[i,j]],spnts[[i,j+1]]}],
 {i,1,n+1},{j,1,m}]}];
maxx=Max[Flatten[Table[Part[spnts[[i,j]],1],{i,1,n+1},{j,1,m+1}]]];
maxy=Max[Flatten[Table[Part[spnts[[i,j]],2],{i,1,n+1},{j,1,m+1}]]];
maxz=Max[Flatten[Table[Part[spnts[[i,j]],3],{i,1,n+1},{j,1,m+1}]]];
axes=Graphics3D[{AbsoluteThickness[1.5],
(*the coordinate axes*)
 Line[{{0,0,maxz},{0,0,0},{maxx,0,0},{0,0,0},{0,maxy,0}}]}];
Show[cpnts,hlines,vlines,axes,patch,PlotRange->All,
 ViewPoint->{2.783,-3.090,1.243}]
```

Figure 13.42. A Rational Bézier Surface Patch.

```
(* A Rational closed Bezier Surface *)
Clear[pwr,bern,spnts,n,m,wt,bzSurf,cpnts,patch,vlines,hlines,axes];
<<:Graphics:ParametricPlot3D.m
r=1; h=3; (* radius & height of cylinder *)
spnts={{{r,0,0},{0,2r,0},{-r,0,0},{0,-2r,0},{r,0,0}},
 {{r,0,h},{0,2r,h},{-r,0,h},{0,-2r,h},{r,0,h}}};
m=Length[spnts[[1]]]-1; n=Length[Transpose[spnts][[1]]]-1;
wt=Table[1, {i,1,n+1},{j,1,m+1}];
pwr[x_,y_]:=If[x==0 && y==0, 1, x^y];
bern[n_,i_,u_]:=Binomial[n,i]pwr[u,i]pwr[1-u,n-i]
bzSurf[u_,w_]:=
 Sum[wt[[i+1,j+1]]spnts[[i+1,j+1]]bern[n,i,u]bern[m,j,w], {i,0,n}, {j,0,m}]/
 Sum[wt[[i+1,j+1]]bern[n,i,u]bern[m,j,w], {i,0,n}, {j,0,m}];
patch=ParametricPlot3D[bzSurf[u,w],{u,0,1}, {w,0,1},
 Compiled->False, DisplayFunction->Identity];
cpnts=Graphics3D[{AbsolutePointSize[4], (* control points *)
```

```
Table[Point[spnts[[i,j]]],{i,1,n+1},{j,1,m+1}]}];
vlines=Graphics3D[{AbsoluteThickness[1],(* control polygon *)
 Table[Line[{spnts[[i,j]],spnts[[i+1,j]]}],{i,1,n},{j,1,m+1}]}];
hlines=Graphics3D[{AbsoluteThickness[1],
 Table[Line[{spnts[[i,j]],spnts[[i,j+1]]}],{i,1,n+1},{j,1,m}]}];
maxx=Max[Flatten[Table[Part[spnts[[i,j]],1],{i,1,n+1},{j,1,m+1}]]];
maxy=Max[Flatten[Table[Part[spnts[[i,j]],2],{i,1,n+1},{j,1,m+1}]]];
maxz=Max[Flatten[Table[Part[spnts[[i,j]],3],{i,1,n+1},{j,1,m+1}]]];
axes=Graphics3D[{AbsoluteThickness[1.5],(* the coordinate axes *)
 Line[{{0,0,maxz},{0,0,0},{maxx,0,0},{0,0,0},{0,maxy,0}}]}];
Show[cpnts,hlines,vlines,axes,patch,PlotRange->All,DefaultFont->{"cmr10",10},
 DisplayFunction->$DisplayFunction,ViewPoint->{0.998,0.160,4.575},Shading->False];
```

Figure Ans.46. A Closed Rational Bézier Surface Patch.

```
for u:=0 step 0.1 to 1 do (* 11 curves *)
 for v:=0 step 0.01 to 1-u do (* 100 pixels per curve *)
  w:=1-u-v;
  Calculate & project point P(u,v,w)
 endfor;
endfor;
```

(In Exercise). Write pseudo-code to draw the three families of curves.

```
(* Triangular Bezier surface patch *)
pnts={{3,3,0}, {2,2,0},{4,2,1}, {1,1,0},{3,1,1},{5,1,2},
 {0,0,0},{2,0,1},{4,0,2},{6,0,3}};
B[i_,j_,k_]:=(n!/(i! j! k!))u^i v^j w^k;
n=3; u=1/6; v=2/6; w=3/6; Tsrpt={0,0,0};
indx:=(n-j)(n-j+1)/2+1+i;
Do[{k=n-i-j, Tsrpt=Tsrpt+B[i,j,k] pnts[[indx]]}, {j,0,n}, {i,0,n-j}];
Tsrpt
```

Figure 13.45. Code for One Point in a Triangular Bézier Patch.

```
(* Triangular Bezier patch by Garry Helzer *)
rules=Solve[{u{a1,b1}+v{a2,b2}+w{a3,b3}=={x,y},u+v+w==1},{u,v,w}]
BarycentricCoordinates[Polygon[{{a1_,b1_},{a2_,b2_},{a3_,b3_}}]] \
 [{x_,y_}]={u,v,w}/.rules//Flatten
Subdivide[l_]:=l/. Polygon[{p_,q_,r_}] :> Polygon /@ \
 ({{p+p,p+q,p+r},{p+q,q+q,q+r},{p+r,q+r,r+r},{p+q,q+r,r+p}}/2)
Transform[F_][L_]:= L /. Polygon[l_] :> Polygon[F /@ l]
P[L_][{u_,v_,w_}]:=
Module[{x,y,z,n=(Sqrt[8Length[L]+1]-3)/2},
 ((List @@ Expand[(x+y+z)^n]) /. {x->u,y->v,z->w}).L]
Param[T_,L_][{x_,y_}]:=With[{p=BarycentricCoordinates[T][{x,y}]},P[L][p]]
```

Run the code below in a separate cell

```
(* Triangular bezier patch for n=3 *)
T=Polygon[{{1, 0}, {0, 1}, {0, 0}}];
L={P300,P210,P120,P030, P201,P111,P021, P102,P012, P003} \
 ={{3,0,0},{2.5,1,.5},{2,2,0},{1.5,3,0},
 {2,0,1},{1.5,1,2},{1,2,.5}, {1,0,1},{.5,1,.5}, {0,0,0}};
SubT=Nest[Subdivide, T, 3];
Patch=Transform[Param[T, L]][SubT];
cpts={PointSize[0.02], Point/@L};
coord={AbsoluteThickness[1],
 Line/@{{{0,0,0},{3.2,0,0}},{{0,0,0},{0,3.4,0}},{{0,0,0},{0,0,1.3}}}};
cpolygon={AbsoluteThickness[2],
 Line[{P300,P210,P120,P030,P021,P012,P003,P102,P201,P300}],
 Line[{P012,P102,P111,P120,P021,P111,P201,P210,P111,P012}]};
Show[Graphics3D[{cpolygon,cpts,coord,Patch}], Boxed->False, PlotRange->All,
 ViewPoint->{2.620, -3.176, 2.236}];
```

Figure 13.46. A Triangular Bézier Surface Patch For $n = 3$.

```
P0300={3,3,0};
P0210={2,2,0}; P1200={4,2,1};
P0120={1,1,0}; P1110={3,1,1}; P2100={5,1,2};
P0030={0,0,0}; P1020={2,0,1}; P2010={4,0,2}; P3000={6,0,3};
n=3; u=1/6; v=2/6; w=3/6;
P0021=u P1020+v P0120+w P0030;
P1011=u P2010+v P1110+w P1020;
P2001=u P3000+v P2100+w P2010;
P0111=u P1110+v P0210+w P0120;
P1101=u P2100+v P1200+w P1110;
P0201=u P1200+v P0300+w P0210;
P0012=u P1011+v P0111+w P0021;
P1002=u P2001+v P1101+w P1011;
P0102=u P1101+v P0201+w P0111;
P0003=u P1002+v P0102+w P0012
B[i_,j_,k_]:=(n!/(i! j! k!))u^i v^j w^k;
P0030 B[0,0,3]+P1020 B[1,0,2]+P2010 B[2,0,1]+P3000 B[3,0,0]+
P0120 B[0,1,2]+P1110 B[1,1,1]+P2100 B[2,1,0]+
P0210 B[0,2,1]+P1200 B[1,2,0]+P0300 B[0,3,0]
```

Figure Ans.48. Triangular Bézier Patch Subdivision Exercise.

```
B={{(1 - a)^3, 3*(-1 + a)^2*a, 3*(1 - a)*a^2, a^3},
  {(-1 + a)^2*(1 - b), (-1 + a)*(-2*a - b + 3*a*b),
   a*(a + 2*b - 3*a*b),
   a^2*b}, {(1 - a)*(-1 + b)^2, (-1 + b)*(-a - 2*b + 3*a*b),
   b*(2*a + b - 3*a*b), a*b^2},
  {(1 - b)^3, 3*(-1 + b)^2*b, 3*(1 - b)*b^2, b^3}};
TB={{(1 - c)^3, (-1 + c)^2*(1 - d), (1 - c)*(-1 + d)^2,
   (1 - d)^3},
  {3*(-1 + c)^2*c, (-1 + c)*(-2*c - d + 3*c*d),
  (-1 + d)*(-c - 2*d + 3*c*d), 3*(-1 + d)^2*d},
  {3*(1 - c)*c^2, c*(c + 2*d - 3*c*d), d*(2*c + d - 3*c*d),
  3*(1 - d)*d^2},
  {c^3, c^2*d, c*d^2, d^3}};
P={{P30,P31,P32,P33},{P20,P21,P22,P23},
 {P10,P11,P12,P13},{P00,P01,P02,P03}};
Q=Simplify[B.P.TB]
```

Code to reparametrize that portion of patch $\mathbf{P}(u,w)$ where $a \le u \le b$.

Chapter 14

```
(* B-spline example of 2 cubic segs and 3 quadr segs for 5 points *)
Clear[Pt,T,t,M3,comb,a,g1,g2,g3];
Pt={{0,0},{0,1},{1,1},{2,1},{2,0}};
(*first,2 cubic segments (dashed)*)
T[t_]:={t^3,t^2,t,1};
M3={{-1,3,-3,1},{3,-6,3,0},{-3,0,3,0},{1,4,1,0}}/6;
comb[i_]:=(T[t].M3)[[i]] Pt[[i+a]];
g1=Graphics[{Red, PointSize[.02],Point/@Pt}];
a=0;
g2=ParametricPlot[comb[1]+comb[2]+comb[3]+comb[4],{t,0,.95},
 PlotRange->All,PlotStyle->{Green,AbsoluteDashing[{5,2}]}];
a=1;
g3=ParametricPlot[comb[1]+comb[2]+comb[3]+comb[4],{t,0.05,1},
 PlotRange->All,PlotStyle->{Green,AbsoluteDashing[{5,2}]}];
(*Now the 3 quadratic segments (solid)*)
T[t_]:={t^2,t,1};
M2={{1,-2,1},{-2,2,0},{1,1,0}}/2;
comb[i_]:=(T[t].M2)[[i]] Pt[[i+a]];
a=0;
```

```
g4=ParametricPlot[comb[1]+comb[2]+comb[3],{t,0,.97}];
a=1;
g5=ParametricPlot[comb[1]+comb[2]+comb[3],{t,0.03,.97}];
a=2;
g6=ParametricPlot[comb[1]+comb[2]+comb[3],{t,0,1}];
Show[g2,g3,g4,g5,g6,g1,PlotRange->All]
```

Figure 14.4. Two Cubic (Dashed) and Three Quadratic (Solid) B-spline Segments.

```
(* Exercise. 8 points, 5-segment uniform B-spline curve, compared to the Bezier
 curve for the same 8 points *)
Clear[p1,p2,p3,p4,p5,bez,l1,g1,g2,g3,g4,g5,g6];
pnts={{1,0},{2,1},{4,0},{4,1}};
p1[t_]:={t^3+6,t^3}/6;
p2[t_]:={3t^2+3t+7,-3t^3+3t^2+3t+1}/6;
p3[t_]:={-3t^3+3t^2+9t+13,4t^3-6t^2+4}/6;
p4[t_]:={2t^3-6t^2+6t+22,-3t^3+6t^2+2}/6;
p5[t_]:={24,t^3-3t^2+3t+5}/6;
bez[t_]:={-3t^3+3t^2+3t+1,4t^3-6t^2+3t};
l1=ListPlot[pnts,PlotStyle->{Red,AbsolutePointSize[6]},
 AspectRatio->Automatic];
g1=ParametricPlot[p1[t],{t,0,.97}];
g2=ParametricPlot[p2[t],{t,0,.97}];
g3=ParametricPlot[p3[t],{t,0,.97}];
g4=ParametricPlot[p4[t],{t,0,.97}];
g5=ParametricPlot[p5[t],{t,0,.97}];
g6=ParametricPlot[bez[t],{t,0,1},
 PlotStyle->{Green,AbsoluteDashing[{5,2}]}];
(*Now the degree-7 Bezier curve*)
pnts={{1,0},{1,0},{1,0},{2,1},{4,0},{4,1},{4,1},{4,1}};
pwr[x_,y_]:=If[x==0&&y==0,1,x^y];
bern[n_,i_,t_]:=Binomial[n,i]pwr[t,i]pwr[1-t,n-i] (*t^i x (1-t)^(n-i)*)
bzCurve[t_]:=Sum[pnts[[i+1]]bern[7,i,t],{i,0,7}]
g7=ParametricPlot[bzCurve[t],{t,0,1},
 PlotStyle->{Blue,AbsoluteDashing[{1,2,2,2}]},AspectRatio->Automatic];
Show[l1,g1,g2,g3,g4,g5,g6,g7,
 PlotRange->All,AspectRatio->Automatic]
```

Figure Ans.51. Comparing a Uniform B-spline and a Bézier Curve for Eight Points.

```
(* Cubic B-spline with tension *)
Clear[t,s,pnts,stnp,tensMat,bsplineTensn,g1,g2,g3,g4];
pnts={{0,0},{0,1},{1,1},{1,0}};
stnp=Transpose[pnts];
tensMat={{2-s,6-s,s-6,s-2},{2s-3,s-9,9-2s,3-s},{-s,0,s,0},{1,4,1,0}};
bsplineTensn[t_]:=Module[{tmpstruc},tmpstruc={t^3,t^2,t,1}.tensMat;
{tmpstruc.stnp[[1]],tmpstruc.stnp[[2]]}/6];
g1=ListPlot[pnts,PlotStyle->{Red,AbsolutePointSize[6]},
 AspectRatio->Automatic];
s=0;
g2=ParametricPlot[bsplineTensn[t],{t,0,1}];
s=3;
g3=ParametricPlot[bsplineTensn[t],{t,0,1},
 PlotStyle->{Green,AbsoluteDashing[{2,2}]}];
s=5;
g4=ParametricPlot[bsplineTensn[t],{t,0,1},
 PlotStyle->{Blue,AbsoluteDashing[{1,2,2,2}]}];
Show[g1,g2,g3,g4,PlotRange->All]
```

Figure 14.7. Cubic B-Spline with Tension.

```
(* B-spline weight functions printed and plotted *)
Clear[bspl,knt,i,k,n,t,p]
bspl[i_,k_,t_]:=If[knt[[i+k]]==knt[[i+1]],0,
(*0<=i<=n*)
 bspl[i,k-1,t] (t-knt[[i+1]])/(knt[[i+k]]-knt[[i+1]])]+If[knt[[i+1+k]]
 ==knt[[i+2]],0,bspl[i+1,k-1,t] (knt[[i+1+k]]-t)/
 (knt[[i+1+k]]-knt[[i+2]])];
bspl[i_,1,t_]:=If[knt[[i+1]]<=t<knt[[i+2]],1,0];
n=4;k=3;(*Note:0<=k<=n*)
```

```
(*knt=Table[i,{i,0,n+k}];*)(*knots for the uniform case*)
knt={0,0,0,1,2,3,3,3};(*knots for the NONuniform case*)
(*Show the weight functions*)
Do[Print["N(",i,",",k,",",t,")=",Simplify[bspl[i,k,t]]],{i,0,n}]
(*Plot them.Plots are separated using .97 instead of 1*)
Do[p[i+1]=Plot[bspl[i,k,t],{t,k-.97,n+.97}],{i,0,n}]
Show[Table[p[i+1],{i,0,n}],Ticks->None,PlotRange->All]
```

Figure 14.16. Code for the B-Spline Weight Functions.

```
(* Plot a B-spline curve. Can also print the weight functions *)
Clear[bspl,knt,i,k,n,t,p,g1,g2,pnt]
 (*First the weight functions*)
bspl[i_,k_,t_]:=If[knt[[i+k]]==knt[[i+1]],0,(*0<=i<=n*)
bspl[i,k-1,t] (t-knt[[i+1]])/
 (knt[[i+k]]-knt[[i+1]])]+If[knt[[i+1+k]]==knt[[i+2]],0,
 bspl[i+1,k-1,t] (knt[[i+1+k]]-t)/(knt[[i+1+k]]-knt[[i+2]])];
bspl[i_,1,t_]:=If[knt[[i+1]]<=t<knt[[i+2]],1,0];
n=4;k=3;
(*Note:0<=k<=n*)(*knt=Table[i,{i,0,n+k}];knots for the uniform case*)
knt={0,0,0,1,2,3,3,3};
(*knots for the open-unif or non-uniform cases*)
(*Do[Print[bspl[i,k,t]],{i,0,n}] Display the weight functions*)
pnt={{0,0},{1,1},{1,2},{2,2},{3,1}};
(*test for n+1=5 control points*)
p[t_]:=Sum[pnt[[i+1]] bspl[i,k,t],{i,0,n}]
(*The curve as a weighted sum*)
g1=ListPlot[pnt,PlotStyle->{Red,AbsolutePointSize[6]},
 AspectRatio->Automatic];
g2=ParametricPlot[p[t],{t,0,.97}];
g3=ParametricPlot[p[t],{t,1,1.97}];
g4=ParametricPlot[p[t],{t,2,3}];
Show[g1,g2,g3,g4,PlotRange->All]
```

Figure 14.17. An Open Uniform B-Spline.

```
(* 8-Point Nonuniform Cubic B-Spline Example. Five Segments *)
Clear[g,Q,pts,seg];
P0={0,0};P1={0,1};P2={1,1};P3={1,0};P4={2,0};
 P5={2.75,1};P6={3,1};P7={3,0};
pts=Graphics[{PointSize[.01],Point/@{P0,P1,P2,P3,P4,P5,P6,P7}}];
seg={AbsoluteDashing[{5,2}],Line[{P1,P2,P3}],Line[{P4,P5,P6,P7}]};
Q[t_]:={((1-t)^3 P0+(3t^3-6t^2+4) P1+(-3t^3+3t^2+3t+1)
 P2+t^3 P3)/6,((2-t)^3 P1+(3t^3-15t^2+21t-5)
 P2+(-3t^3+12t^2-12t+4) P3+(t-1)^3 P4)/6,((3-t)^3
 P2+(3t^3-24t^2+60t-44) P3+(-3t^3+21t^2-45t+31)
 P4+(t-2)^3 P5)/6,((4-t)^3 P3+(3t^3-33t^2+117t-131)
 P4+(-3t^3+30t^2-96t+100) P5+(t-3)^3 P6)/6,((5-t)^3
 P4+(3t^3-42t^2+192t-284) P5+(-3t^3+39t^2-165t+229)
 P6+(t-4)^3 P7)/6};
g=Table[ParametricPlot[Q[t][[i]],{t,i-1,0.97i}],{i,1,5}];
Show[g,pts,Graphics[seg],PlotRange->All]
```

For the four segments of part (b), the only difference is

```
Q[t_]:={(1-t)^3/6 P0 +(11t^3-15t^2-3t+7)/12 P1+(-5t^3+3t^2+3t+1)/4
P2 +t^3/2 P3,
(2-t)^3/2 P2 +(5t^3-27t^2+45t-21)/4 P3+(-11t^3+51t^2-69t+29)/12
P4 +(t-1)^3/6 P5,
(3-t)^3/4 P3 +(7t^3-57t^2+147t-115)/12 P4+(-3t^3+21t^2-45t+31)/6
P5 +(t-2)^3/6 P6,((4-t)^3
P4 +(3t^3-33t^2+117t-131)P5+(-3t^3+30t^2-96t+100)P6
+(t-3)^3 P7)/6};
g=Table[ParametricPlot[Q[t][[i]], {t,i-1,0.97i}], {i,1,4}];
```

For the three segments of part (c), the only difference is

```
Q[t_]:={(1-t)^3 P0 /6+(11t^3-15t^2-3t+7)P1 /12+(-7t^3+3t^2+3t+1)P2
 /4+t^3 P3,(2-t)^3 P3+(7t^3-39t^2+69t-37)P4
 /4+(-11t^3+51t^2-69t+29) P5 /12+(t-1)^3 P6
 /6,(3-t)^3 P4 /4+(7t^3-57t^2+147t-115)P5
```

```
/12+(-3t^3+21t^2-45t+31) P6 /6+(t-2)^3 P7 /6};
g=Table[ParametricPlot[Q[t][[i]],{t,i-1,0.97i}],{i,1,3}];
```

For the two segments of part (d), the only difference is

```
Q[t_]:={(1-t)^3P0 /6 +(11t^3-15t^2-3t+7)P1
/12+(-7t^3+3t^2+3t+1)P2
/4 +t^3 P3,(2-t)^3 P4 +(7t^3-39t^2+69t-37)P5
/4+(-11t^3+51t^2-69t+29)P6 /12+(t-1)^3P7 /6};
g=Table[ParametricPlot[Q[t][[i]],{t,i-1,0.97i}],{i,1,2}];
```

Figure 14.18. Code for an 8-Point Nonuniform B-Spline Example, Figure 14.19.

```
(* Compute the nonuniform weight functions for the 8-point example that follows *)
Clear[bspl,knt]
bspl[i_,k_,t_]:=If[knt[[i+k]]==knt[[i+1]],0, (* 0<=i<=n *)
 bspl[i,k-1,t] (t-knt[[i+1]])/(knt[[i+k]]-knt[[i+1]])] \
 +If[knt[[i+1+k]]==knt[[i+2]],0,
 bspl[i+1,k-1,t] (knt[[i+1+k]]-t)/(knt[[i+1+k]]-knt[[i+2]])];
bspl[i_,1,t_]:=If[knt[[i+1]]<=t<knt[[i+2]], 1, 0];
n=4; k=4; (* Note: 0<=k<=n *)
knt={-3,-2,-1,0,1,2,3,4,5,6,7,8}; (* knots for nonuniform case *)
bspl[i,k,t] (* assign a value to i *)
```

Figure 14.20. Eight-Point Nonuniform B-Spline Example; Code for Blending Functions.

```
(* Rational B-spline example. w_2=0, .5, 1, 5 (Slow!) *)
Clear[bspl,knt,w,pnts,cur1,cur2,cur3,cur4,R] (*weight functions*)
bspl[i_,k_,t_]:=If[knt[[i+k]]==knt[[i+1]],0,(*0<=i<=n*)
 bspl[i,k-1,t] (t-knt[[i+1]])/
  (knt[[i+k]]-knt[[i+1]])]+If[knt[[i+1+k]]==knt[[i+2]],0,
 bspl[i+1,k-1,t] (knt[[i+1+k]]-t)/(knt[[i+1+k]]-knt[[i+2]])];
bspl[i_,1,t_]:=If[knt[[i+1]]<=t<knt[[i+2]],1,0];
R[i_,t_]:=(w[[i+1]] bspl[i,k,t])/Sum[w[[j+1]] bspl[j,k,t],{j,0,n}];
n=4;k=3;w={1,1,0,1,1};(*weights*)knt={0,0,0,1,2,3,3,3};(*knots*)
pnts={{0,0},{0,1},{1,0},{2,1},{2,0}};
cur1=ParametricPlot[Sum[(R[i,t] pnts[[i+1]]),{i,0,n}],{t,0,3}];
w[[3]]=0.5;
cur2=ParametricPlot[Sum[(R[i,t] pnts[[i+1]]),{i,0,n}],{t,0,3}];
w[[3]]=1;
cur3=ParametricPlot[Sum[(R[i,t] pnts[[i+1]]),{i,0,n}],{t,0,3}];
w[[3]]=5;
cur4=ParametricPlot[Sum[(R[i,t] pnts[[i+1]]),{i,0,n}],{t,0,3}];
g1=ListPlot[pnts,PlotStyle->{Red,AbsolutePointSize[6]},
 AspectRatio->Automatic];
Show[g1,cur1,cur2,cur3,cur4,PlotRange->All]
```

Figure 14.21. Effects of Varying Weight w_2.

```
(*One third of a circle done by rational B-spline*)
P0={0,-1};P1={-1.732,-1};P2={-0.866,0.5};w1=0.5;
pnts=ListPlot[{P0,P1,P2},PlotStyle->{Red,AbsolutePointSize[6]}];
axs={AbsoluteThickness[1],Line[{P0,P1,P2}]};
th=ParametricPlot[((1-t)^2 P0+2w1 t (1-t)P1+t^2 P2)/
 ((1-t)^2+2w1 t (1-t)+t^2),{t,0,1}];
Show[Graphics[axs],th,pnts,PlotRange->All]
```

Figure 14.23. Control Points for Circles.

```
(* BiQuadratic B-spline Patch Example *)
Clear[T,Pnts,Q,comb,g1,g2];
T[t_]:={t^2,t,1};
Pnts={{{0,0,0},{0,1.5,0},{0,2,0}},{{1,0,0},
 {1,1,1},{1,2,0}},{{2,0,0},{2,0.5,0},{2,2,0}}};
Q={{1,-2,1},{-2,2,0},{1,1,0}};
g1=Graphics3D[{Red,AbsolutePointSize[6],
 Table[Point[Pnts[[i,j]]],{i,1,3},{j,1,3}]}];
comb[i_]:=((1/4)T[u].Q.Pnts)[[i]] (Transpose[Q].T[w])[[i]]
g2=ParametricPlot3D[comb[1]+comb[2]+comb[3],
```

```
{u,0,1},{w,0,1},AspectRatio->Automatic,
 Ticks->{{0,1,2},{0,1,2},{0,1}}];
Show[g2,g1,ViewPoint->{-0.196,-4.177,1.160},PlotRange->All]
```

Figure 14.27. A Biquadratic B-Spline Surface Patch.

```
0 0 0  0 1 1  0 2 1  0 2 0
1 0 0  1 1 2  1 2 1  1 3 2
2 0 0  2 1 3  2 2 2  2 3 3
3 0 0  3 1 2  3 2 1  3 3 2
4 0 0  4 1 1  4 2 1  4 2 0
```

```
(*a general uniform B-spline surface patch*)
bspl[i_,k_,t_]:=bspl[i,k-1,t] (t-knt[[i+1]])/
 (knt[[i+k]]-knt[[i+1]])+bspl[i+1,k-1,t] (knt[[i+1+k]]-t)/
 (knt[[i+1+k]]-knt[[i+2]]) (*0<=i<=n*)
bspl[i_,1,t_]:=If[knt[[i+1]]<=t<knt[[i+2]],1,0];
n=3;kn=3;m=4;km=3;(*Note:0<=kn<=n 0<=km<=m*)
knt=Table[i,{i,0,m+km}];
(*uniform knots*)(*Input triplets from data file*)
surpnts=ReadList["surf.pnts",{Number,Number,Number},
 RecordLists->True];
bsplSurf[u_,w_]:=Sum[Sum[surpnts[[i+1,j+1]]
 bspl[i,km,u],{i,0,m}]bspl[j,kn,w],{j,0,n}]
g1=Graphics3D[{Red,AbsolutePointSize[6],
 Table[Point[surpnts[[i,j]]],{i,1,5},{j,1,4}]}];
g2=ParametricPlot3D[bsplSurf[u,w],{u,km-1,m+1},{w,kn-1,n+1},
 AspectRatio->Automatic];
Show[g1,g2,PlotRange->All,ViewPoint->{1.389,-3.977,1.042}]
```

Figure 14.29. A Quadratic-Cubic B-Spline Surface Patch.

Chapter 15

```
(* Chaikin algorithm for a control polygon *)
n=4;
(*p={p0,p1,p2,p3,p4,p5};*)
p={{0,0},{0,4},{3,4},{4,0},{6,6}};
Show[Graphics[Line[p]]]
q=Table[If[OddQ[i],
(*then*){(3p[[i]]+p[[i+1]])/4,(p[[i]]+3p[[i+1]])/4},
(*else*){(3p[[i]]+p[[i+1]])/4,(p[[i]]+3p[[i+1]])/4}],{i,1,n}];
q=Flatten[q,1]
Show[Graphics[{Green,AbsoluteDashing[{5,2}],
 Line[p]}],Graphics[Line[q]],PlotRange->All]
r=Table[If[OddQ[i],
 (*then*){(3q[[i]]+q[[i+1]])/4,(q[[i]]+3q[[i+1]])/4},
 (*else*){(3q[[i]]+q[[i+1]])/4,(q[[i]]+3q[[i+1]])/4}],{i,1,2n-1}];
r=Flatten[r,1]
Show[Graphics[{Green,AbsoluteDashing[{2,2}],
 Line[p]}],Graphics[Line[r]],PlotRange->All]
```

Figure 15.5. Chaikin's Algorithm for a Control Polygon.

```
a={{4,4,0},{1,6,1},{0,4,4}}/8; {p10,p11,p12}=a.{p00,p01,p02};
{p12,p13,p14}=a.{p01,p02,p03}; {p20,p21,p22}=a.{p10,p11,p12};
{p22,p23,p24}=a.{p11,p12,p13}; {p24,p25,p26}=a.{p12,p13,p14};
{p30,p31,p32}=a.{p20,p21,p22}; {p32,p33,p34}=a.{p21,p22,p23};
{p34,p35,p36}=a.{p22,p23,p24}; {p36,p37,p38}=a.{p23,p24,p25};
{p38,p39,p310}=a.{p24,p25,p26}; Simplify[(p36+4 p37+p38)/6]
```

Figure Ans.52. Code for Exercise 15.6

```
(* reparametrize biquadratic B-spline surface *)
Clear[a,b,c,d,A,B,TB,H,M,P,Q];
M={{1,-2,1},{-2,2,0},{1,1,0}}/2;
```

```
A={{(b-a)^2,0,0},{2a(b-a),b-a,0},{a^2,a,1}};
(* B=MatrixForm[Simplify[Inverse[M].A.M]] *)
B={{((1 - a)*(1 - 2*a + b))/2, (1 + 3*a - 4*a^2 - b + 2*a*b)/2,
  a^2 - (a*b)/2}, {1/2 - a/2 - b/2 + (a*b)/2, (1 + a + b - 2*a*b)/2,
  (a*b)/2}, {((1 + a - 2*b)*(1 - b))/2, (1 - a + 3*b + 2*a*b - 4*b^2)/2,
  -(a*b)/2 + b^2}};
TB={{((1 - c)*(1 - 2*c + d))/2, 1/2 - c/2 - d/2 + (c*d)/2,
  ((1 + c - 2*d)*(1 - d))/2},
 {(1 + 3*c - 4*c^2 - d + 2*c*d)/2, (1 + c + d - 2*c*d)/2,
  (1 - c + 3*d + 2*c*d - 4*d^2)/2},
 {c^2 - (c*d)/2, (c*d)/2, -(c*d)/2 + d^2}};
P={{P00,P01,P02},{P10,P11,P12},{P20,P21,P22}};
Q=Simplify[B.P.TB]
a=0; b=.5; c=0; d=.5; Q
```

Figure 15.9. Code for the Nine Control Points of the “Upper-Left” Patch.

```
(* reparametrize bicubic B-spline surface *)
Clear[a,b,c,d,A,B,TB,H,M,P,Q];
M={{-1,3,-3,1},{3,-6,3,0},{-3,0,3,0},{1,4,1,0}}/6;
A={{(b-a)^3,0,0,0},{3a(b-a)^2,(b-a)^2,0,0},{3a^2(b-a),2a(b-a),b-a,0},{a^3,a^2,a,1}};
(*B=Simplify[Inverse[M].A.M] *)
B={{((1 - a)*(1 - 5*a + 6*a^2 + 3*b - 7*a*b + 2*b^2))/6,
  (4 - 22*a^2 + 18*a^3 + 20*a*b - 21*a^2*b - 4*b^2 + 6*a*b^2)/6,
  1/6 + a + (11*a^2)/6 - 3*a^3 - b/2 - (5*a*b)/3 + (7*a^2*b)/2 + b^2/3 -
   a*b^2, a^3 - (7*a^2*b)/6 + (a*b^2)/3},
 {((-1 + a)*(-1 + 2*a - 2*a*b + b^2))/6,
  (4 - 4*a^2 - 4*a*b + 6*a^2*b + 2*b^2 - 3*a*b^2)/6,
  1/6 + a/2 + a^2/3 + (a*b)/3 - a^2*b - b^2/6 + (a*b^2)/2,
  (a*(2*a - b)*b)/6}, {((-1 + a)*(1 + a - 2*b)*(-1 + b))/6,
  (4 + 2*a^2 - 4*a*b - 3*a^2*b - 4*b^2 + 6*a*b^2)/6,
  1/6 - a^2/6 + b/2 + (a*b)/3 + (a^2*b)/2 + b^2/3 - a*b^2,
  (a*b*(-a + 2*b))/6}, {((1 - b)*(1 + 3*a + 2*a^2 - 5*b - 7*a*b + 6*b^2))/
   6, (4 - 4*a^2 + 20*a*b + 6*a^2*b - 22*b^2 - 21*a*b^2 + 18*b^3)/6,
  1/6 - a/2 + a^2/3 + b - (5*a*b)/3 - a^2*b + (11*b^2)/6 + (7*a*b^2)/2 -
   3*b^3, (a^2*b)/3 - (7*a*b^2)/6 + b^3}};
TB={{((1 - a)*(1 - 5*a + 6*a^2 + 3*b - 7*a*b + 2*b^2))/6,
  ((-1 + a)*(-1 + 2*a - 2*a*b + b^2))/6,
  ((-1 + a)*(1 + a - 2*b)*(-1 + b))/6,
  ((1 - b)*(1 + 3*a + 2*a^2 - 5*b - 7*a*b + 6*b^2))/6},
 {(4 - 22*a^2 + 18*a^3 + 20*a*b - 21*a^2*b - 4*b^2 + 6*a*b^2)/6,
  (4 - 4*a^2 - 4*a*b + 6*a^2*b + 2*b^2 - 3*a*b^2)/6,
  (4 + 2*a^2 - 4*a*b - 3*a^2*b - 4*b^2 + 6*a*b^2)/6,
  (4 - 4*a^2 + 20*a*b + 6*a^2*b - 22*b^2 - 21*a*b^2 + 18*b^3)/6},
 {1/6 + a + (11*a^2)/6 - 3*a^3 - b/2 - (5*a*b)/3 + (7*a^2*b)/2 +
   b^2/3 - a*b^2, 1/6 + a/2 + a^2/3 + (a*b)/3 - a^2*b - b^2/6 +
   (a*b^2)/2, 1/6 - a^2/6 + b/2 + (a*b)/3 + (a^2*b)/2 + b^2/3 - a*b^2,
  1/6 - a/2 + a^2/3 + b - (5*a*b)/3 - a^2*b + (11*b^2)/6 + (7*a*b^2)/2 -
   3*b^3}, {a^3 - (7*a^2*b)/6 + (a*b^2)/3, (a*(2*a - b)*b)/6,
  (a*b*(-a + 2*b))/6, (a^2*b)/3 - (7*a*b^2)/6 + b^3}};
P={{P30,P31,P32,P33},{P20,P21,P22,P23},{P10,P11,P12,P13},{P00,P01,P02,P03}};
Q=Simplify[B.P.TB]
a=0; b=.5; c=0; d=.5; Q
```

Figure 15.13. Code for the 16 Control Points of the “Uper-Left” Patch.

Chapter 16

```
(* 2 sweep surface examples *)
alf=1;
ParametricPlot3D[{u Cos[2Pi w],u Sin[2Pi w],alf w},{u,0,1},{w,0,1},
 ViewPoint->{3.369,-2.693,0.479},PlotPoints->20]
m={-3u^3+6u^2+3u,-3u^3+3u^2+1,3u^2-3u+1,1}.
 {{1,0,0,0},{0,1,0,0},{0,0,1,0},{-4w^3+3w^2+3w,-6w^2+6w,-2w^3+3w,1}};
ParametricPlot3D[Drop[m,-1],{u,0,1},{w,0,1},
 ViewPoint->{4.068,-1.506,0.133},PlotPoints->20]
```

Figure 16.1. Two Sweep Surfaces.

```
ParametricPlot3D[{3u,Sin[w],w}, {u,0,1},{w,0,4Pi},
```

```
Ticks->False, AspectRatio->Automatic]
```

Figure Ans.53. A Sweep Surface.

```
(* Mobius strip as a sweep surface *)
Clear[r,roty,rotz,segm];
segm[t_]:={t,0,0}; (* a short line segment *)
roty[phi_]:={{Cos[phi],0,-Sin[phi]},{0,1,0},{Sin[phi],0,Cos[phi]}};
rotz[phi_]:={{Cos[phi],-Sin[phi],0},{Sin[phi],Cos[phi],0},{0,0,1}};
ParametricPlot3D[Evaluate[rotz[phi].(roty[phi/2].segm[t]+{20,0,0})],
 {phi,0,2Pi}, {t,-3,3}, Boxed->True, PlotPoints->{35,2}, Axes->False]
Show[{%,Graphics3D[{AbsoluteThickness[1], (* show the 3 axes *)
 Line[{{0,0,30},{0,0,0},{30,0,0},{0,0,0},{0,30,0}}]}]},
 PlotRange -> All]
```

Figure 16.2. A Möbius Strip.

```
(* Sweep surface example. Lofted surface with scaling transform *)
pnts={{-1,-1,0},{1,-1,0},{-1,1,0},{0,1,1},{1,1,0}};
{2u-1,2w-1,4u w (1-u)}.{{w,0,0},{0,1,0},{0,0,1}};
g1=ParametricPlot3D[%,{u,0,1},{w,0,1},AspectRatio->Automatic,
 Ticks->{{0,1},{0,1},{0,1}}];
g2=Graphics3D[{Red,AbsolutePointSize[6],Table[Point[pnts[[i]]],{i,1,5}]}];
Show[g1,g2,ViewPoint->{-0.139,-1.179,1.475},PlotRange->All]
```

Figure 16.3. A Lofted Swept Surface.

```
(* A Sweep Surface. Curve Cu[u,w] times matrix Trn[w] *)
Clear[Cu,Trn];
Cu[u_,w_]:={u,1,u+2}w+{-u,1,u-2}(1-w);
Trn[w_]:={{Cos[2Pi w],Sin[2Pi w],0},{-Sin[2Pi w],
 Cos[2Pi w],0},{0,0,1}};
ParametricPlot3D[{Cu[u,w].Trn[w][[1]],Cu[u,w].
 Trn[w][[2]],Cu[u,w].Trn[w][[3]]},{u,0,1},{w,0,1},
 Ticks->None,PlotRange->All,
 AspectRatio->Automatic,ViewPoint->{-0.510,-1.365,1.210}]
```

Figure 16.4. Sweeping while Rotating.

```
R=10; r=2;   (* The Torus as a surface of revolution *)
ParametricPlot3D[
 {(R+r Cos[2Pi u])Cos[2Pi w],-(R+r Cos[2Pi u])Sin[2Pi w],
 r Sin[2Pi u]},{u,0,1},{w,0,1},
 ViewPoint->{-0.028, -4.034, 1.599}]
```

Figure Ans.54. The Torus as a Surface of Revolution.

```
(* A Chalice *)
(*the profile*)
ParametricPlot[{.5u^3-.3u^2-.5u-.2,u+1},{u,-1,1},
 AspectRatio->Automatic]
(*the surface*)
RevolutionPlot3D[{.5u^3-.3u^2-.5u-.2,u+1},{u,-1,1},
 PlotPoints->40]
```

Figure 16.8. A Chalice as a Surface of Revolution.

```
(*Surface of revolution*)
Clear[basis,Cubi];
(*as a combination of 2 cubic B-splines*)
(*matrix 'basis' has dimensions 4x4x3*)
basis={{{0,0,0},{0,-3/2,0},{0,-3/2,3},{0,0,3}},
{{0,0,0},{-3/2,0,0},{-3/2,0,3},{0,0,3}},
{{0,0,0},{0,3/2,0},{0,3/2,3},{0,0,3}},
```

```
{{0,0,0},{3/2,0,0},{3/2,0,3},{0,0,3}}};
Cubi={{-1,3,-3,1},{3,-6,3,0},{-3,0,3,0},{1,4,1,0}};
prt[i_]:=basis[[Range[1,4],Range[1,4],i]];
(*'prt' extracts component i from the 3rd dimen of 'basis'*)
coord[i_]:={u^3,u^2,u,1}.Cubi.prt[i].Transpose[Cubi].{w^3,w^2,w,1};
ParametricPlot3D[{coord[1],coord[2],coord[3]}/36,{u,0,1},{w,0,1},
 Prolog->AbsoluteThickness[.5],ViewPoint->{1.736,-0.751,-0.089}]
```

Figure 16.10. A Quarter-Circle Surface of Revolution made of B-Splines.

Chapter 17

```
g1=Plot[{Red,Cos[t]},{t,-Pi/2,Pi/2}];
g2=Plot[Cos[t]^5,{t,-Pi/2,Pi/2}];
g3=Plot[Cos[t]^10,{t,-Pi/2,Pi/2}];
g4=Plot[Cos[t]^50,{t,-Pi/2,Pi/2}, PlotRange->All];
Show[g1,g2,g3,g4,PlotRange->All]
```

Figure 17.7. The Behavior of $\cos^n\theta$.

```
procedure Gouraud(P1,P2,P3,I1,I2,I3);
real I; point P;
for u:=0 to 1 step 0.1 do
 for w:=0 to 1-u step 0.001 do
  I:=I1*(1-u-w)+I2*u+I3*w;
  P:=P1*(1-u-w)+P2*u+P3*w;
  Pixel(P,I);
end;
```

Figure 17.10. Scanning a Triangle.

```
procedure Gouraud4(P1,P2,P3,P4,I1,I2,I3,I4);
real I, Ia, Ib; point P, Pa, Pb;
for u:=0 to 1 step 0.1 do
Ia:=I2*(1-u)+I1*u; Ib:=I3*(1-u)+I4*u;
Pa:=P2*(1-u)+P1*u; Pb:=P3*(1-u)+P4*u;
 for w:=0 to 1-u step 0.001 do
  I:=Ia*(1-w)+Ib*w;
  P:=Pa*(1-w)+Pb*w;
  Pixel(P,I);
end;
```

Figure Ans.57. Scan Procedure for a Four-Sided Polygon.

Chapter 19

```
p0={0,1}; p1={5,1}; p2={5,0}; p3={4,.5};
Bez[t_]:=(1-t)^3p0+3t(1-t)^2p1+3t^2(1-t)p2+t^3p3;
tbl=Table[Bez[t], {t,0,1,.01}];
(* tab1 is a list of lengths of straight segments *)
tab1=Table[Sqrt[(tbl[[i+1,1]]-tbl[[i,1]])^2
 +(tbl[[i+1,2]]-tbl[[i,2]])^2], {i,1,100}];
(* tab2 is a list of accumulated lengths *)
tab2={tab1[[1]]};
Do[tab2=Append[tab2,tab1[[i]]+tab2[[i-1]]],{i,2,100}];
tab2=tab2/tab2[[100]]; (* normalize tab2 *)
```

```
tab3={0}; d=.1;
(* tab3 is a list of non-equally-spaced parameter values *)
Do[If[tab2[[i]]>d, {tab3=Append[tab3,i/100], d=d+.1}], {i,1,100}];
tab3=Append[tab3,1];
len=Length[tab3];
tab4=Table[Bez[tab3[[i]]], {i,1,len}];
 (* use tab3 as the parameter values *)
ListPlot[tab4] (* display equally-spaced points *)
ListPlot[tbl]  (* display 101 non-equally-spaced points *)
```

Figure 19.2. Normalized Accumulated Arc Lengths.

```
#include <stdio.h>
#include <math.h> // for function fabs
float totl_arc; // global variable
void Add_tabl(float, float);
float Gauss(float, float);
float Subdivide(float left, float right, float full_intr, float eps){
float mid, left_arc, right_arc, left_sub;
 mid=(left+right)/2;
 left_arc=Gauss(left,mid);
 right_arc=Gauss(mid,right);
 if(fabs(full_intr-left_arc-right_arc)<eps)
  {left_sub=Subdivide(left,mid,left_arc,eps/2.0);
  totl_arc=totl_arc+left_sub;
  Add_tabl(mid,totl_arc);
  return(Subdivide(mid,right,right_arc,eps/2.0)+left_sub);}
 else
  return(left_arc+right_arc);
}
int main(){
float left, right, full_intr, eps;
left=0; right=1.0; totl_arc=0; eps=0.001;
 full_intr=Gauss(left,right);
 Subdivide(left,right,full_intr,eps);
 }
```

Figure 19.3. Procedure Subdivide.

```
(* Two interpolations of vectors with 90 deg *)
d1={1,0}; d2={0,1};
(* Generate 11 linearly interpolated vectors in 'vec' *)
vec=Table[(1-t)d1+t d2,{t,0,1,.1}];
(* Normalize these vectors *)
Do[vec[[i]]=vec[[i]]/Sqrt[vec[[i,1]]^2+vec[[i,2]]^2], {i,1,11}];
(* Show them *)
Table[ArcCos[vec[[1]].vec[[i+1]]]/Degree, {i,1,10}]
Table[ArcCos[vec[[i]].vec[[i+1]]]/Degree, {i,1,10}]
(* Generate 11 spherically interpolated vectors in 'vec' *)
vec=Table[(Sin[90(1-t)Degree]d1+Sin[90t Degree]d2),{t,0,1,.1}];
(* Normalize these vectors *)
Do[vec[[i]]=vec[[i]]/Sqrt[vec[[i,1]]^2+vec[[i,2]]^2], {i,1,11}];
(* Show them *)
Table[ArcCos[vec[[1]].vec[[i+1]]]/Degree, {i,1,10}]
```

```
Table[ArcCos[vec[[i]].vec[[i+1]]]/Degree, {i,1,10}]
```

Code for Table 19.7.

```
Clear[T];p1=0;p2=1;(*Quadratic Blending*)
T[pw_]:=Plot[(1-t)^2 p1+2t (1-t)pw+t^2 p2,{t,0,1},
 PlotStyle->{Red, AbsoluteThickness[.5]}];
Show[T[0],T[.25],T[.5],T[.75],T[1],
 PlotRange->All,AspectRatio->Automatic]

Clear[Bez];p0=0;p3=1;(*Bezier Blending*)Bez[p1_,p2_]:=
Plot[(1-t)^3 p0+3t (1-t)^2p1+3t^2(1-t)p2+t^3 p3,{t,0,1},
 AspectRatio->Automatic,PlotStyle->{Red, AbsoluteThickness[.5]}];
Show[Bez[0,.1],Bez[.2,.3],Bez[.333,.667],Bez[.7,.8],Bez[.9,1],
 PlotRange->All]
```

Figure 19.15. (a) Quadratic Blending. (b) Cubic Blending. (c) Code.

```
Clear[T,H,Hermi]; (* Hermite Interpolation *)
T={t^3,t^2,t,1};
H={{2,-2,1,1},{-3,3,-2,-1},{0,0,1,0},{1,0,0,0}};
(*B={0,1,0,0};*)
Hermi[v1_,v2_,s_,e_]:=Plot[T.H.{v1,v2,s,e},{t,0,1},
 AspectRatio->Automatic, Prolog->AbsoluteThickness[.4]];
Show[Hermi[0,1,0,0], Hermi[0,1,1,1], Hermi[0,1,2,2],
 Hermi[0,1,3,3], Hermi[0,1,4,4]]
```

Figure 19.16. Hermite Interpolation.

```
Clear[fa,fb,fm,den,a,b];
a=.1; b=.3;
fa:=2a(Sin[Pi(t-a)/(2a)]+1)/Pi;
fb:=Sin[Pi(t-b)/(2(1-b))]2(1-b)/Pi+2a/Pi+b-a;
fm:=2a/Pi+t-a;
den=2a/Pi+2(1-b)/Pi+b-a;
T:=If[t<a,fa/den,If[t>b,fb/den,fm/den]];
Plot[T, {t,0,1}, AspectRatio->1]
```

Figure 19.17. Ease-in/Ease-out with a Sine Function.

Chapter 20

```
DATA XPL /2, 4, 6, 8, 10 /
DATA YPL /1, 5, 2, 6,  4/
POLYLINE(5, XPL, YPL)
```

Polyline Example.

```
glBegin(GL_POINTS);
glVertex2f( 1.0, 1.0 );
glVertex2f( 2.0, 1.0 );
glEnd();

glBegin(GL_TRIANGLES);
```

```
glColor3f( 1.0, 0.0, 0.0 );
glVertex3f( 0.3, 1.0, 0.5 );
glVertex3f( 2.7, 0.85, 0.0 );
glVertex3f( 2.7, 1.15, 0.0 );
glEnd();
```

Examples of OpenGL Groups.

```
-6
18
add

newpath
72 144 moveto
216 72 lineto
stroke
showpage

newpath
288 288 moveto
0 72 rlineto
72 0 rlineto
0 -72 rlineto
-72 0 rlineto
4 setlinewidth
stroke showpage

/square
{newpath
moveto
0 72 rlineto
72 0 rlineto
0 -72 rlineto
closepath}
def

72 144 square stroke
288 288 square 4 setlinewidth stroke
0 288 square .5 setgray fill
showpage
```

Examples of PostScript Codes.

```
function PaethPredictor (a, b, c)
begin
; a=left, b=above, c=upper left
p:=a+b-c ;initial estimate
pa := abs(p-a) ; compute distances
pb := abs(p-b) ; to a, b, c
pc := abs(p-c)
; return nearest of a,b,c,
; breaking ties in order a,b,c.
if pa<=pb AND pa<=pc then return a
else if pb<=pc then return b
```

```
else return c
end
```

A Paeth Filter

Chapter 22

```
(* Fractalization of a line *)
st = 1; en = 129;
lin = Table[{0, 0}, {i, en}];
lin[[1]] = {3., 45};
lin[[en]] = {120., 67};
frac[s_, e_] := Module[{mid, t},
 mid = (s + e)/2;
 t = RandomReal[NormalDistribution[0, .15]];
 lin[[mid]]={(lin[[s,1]]+lin[[e,1]])/2-(lin[[e,2]]-lin[[s,2]])t,
 (lin[[s,2]]+lin[[e,2]])/2+(lin[[e,1]]-lin[[s,1]])t};
 If[(e-s)>2, {frac[s, mid]; frac[mid, e]}]];
frac[st, en];
Graphics[Line[lin]]
```

Figure 22.5. A Fractured Line.

```
procedure FracRecur(grid);
 Compute the center point in a diamond step.
 Compute the edge points in a square step.
 If the grid is more than 3 by 3,
  invoke FracRecur recursively four times, with the
  addresses of the four quarters of the grid

Main()
Input values for the four corners.
invoke FracRecur with the grid address.
end.
```

Recursive Algorithm to Fractalize a Line.

```
Tn={{{.5,0},{0,.5}},{{.5,0},{0,.5}},{{.5,0},{0,.5}}};
Mr={{0,0},{.25,0.259808},{.5,0}};
pnt={{0,0}};
Do[{r=RandomInteger[{1, 3}],
  pnt=Append[pnt, pnt[[i]].Tn[[r]]+Mr[[r]]]},{i,5000}]
ListPlot[pnt, Axes->False, PlotStyle->{Blue}]
```

Figure 22.10. Sierpinski Triangle in IFS.

```
(* IFS  for a fern *)
Tn={{{.16,0},{0,0}},{{.85,.04},{-0.04,0.85}},
 {{0.22,-0.26},{0.23,0.2}},{{0.24,0.28},
 {0.26,-0.15}}};
Mr={{0,0},{0,1.6},{0,1.6},{0,0.44}};
pnt={{0,0}};
rc=Flatten[{1, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4,
    Table[2, {i, 85}]}];
Do[{r=RandomChoice[rc],
  pnt=Append[pnt, pnt[[i]].Tn[[r]]+Mr[[r]]]}, {i,1000}]
ListPlot[pnt, Axes->False, PlotStyle->{Red}]
```

Figure 22.11. A Fern in IFS.

```
(* A nonlinear dynamical system. A recurrence relation *)
```

```
r=4.2;
rpt=30; (* # of computation steps *)
ar=Table[0, {i, rpt}];
ar[[1]]=0.6; (* Initial value *)
Do[{ar[[i+1]]=r ar[[i]] (1-ar[[i]])}, {i, rpt-1}]
ar
```

Code to Experiment with Attractors.

```
x:=0.0;
for i:=1 to 12 do x:=x+Rnd();
Gauss:=x-6.0;
```

Produce Gaussian random numbers.

Chapter 23

```
fpc = OpenRead["test.txt"];
g = 0; ar = Table[{i, 0}, {i, 256}];
While[0 == 0,
 g = Read[fpc, Byte];
 (* Skip space, newline & backslash *)
 If[g==10||g==32||g==92, Continue[]];
 If[g==EndOfFile, Break[]];
 ar[[g, 2]]++] (* increment counter *)
Close[fpc];
ar = Sort[ar, #1[[2]] > #2[[2]] &];
tot = Sum[
ar[[i,2]], {i,256}] (* total chars input *)
Table[{FromCharacterCode[ar[[i,1]]],ar[[i,2]],ar[[i,2]]/N[tot,4]},
 {i,93}] (* char code, freq., percentage *)
TableForm[%]
```

Figure 23.3. Code for Table 23.2.

```
rm=RandomReal[1, {32,32}];
Graphics[Raster[rm]]
irm=Inverse[rm];
Graphics[Raster[irm,Automatic, {Min[irm],Max[irm]}]]
```

Figure 23.8. Maps of (a) a Random Matrix and (b) its Inverse.

```
a=rand(32); b=inv(a);
figure(1), imagesc(a), colormap(gray); axis square
figure(2), imagesc(b), colormap(gray); axis square
figure(3), imagesc(cov(a)), colormap(gray); axis square
figure(4), imagesc(cov(b)), colormap(gray); axis square
 Mathematica code
rm=RandomReal[1,{32,32}];
Graphics[Raster[rm]]
arm=Covariance[rm];
Graphics[Raster[arm,Automatic,{Min[arm],Max[arm]}]]
irm=Inverse[rm];
Graphics[Raster[irm,Automatic,{Min[irm],Max[irm]}]]
brm=Covariance[irm];
Graphics[Raster[brm,Automatic,{Min[brm],Max[brm]}]]
```

Figure Ans.60. Covariance Matrices of Correlated and Decorrelated Values.

```
function b=rgc(a,i)
[r,c]=size(a);
b=[zeros(r,1),a; ones(r,1),flipud(a)];
if i>1, b=rgc(b,i-1); end;
```

Table 23.11. First 32 Binary and Reflected Gray Codes.

```
a=linspace(0,31,32); b=bitshift(a,-1);
b=bitxor(a,b); dec2bin(b)
```

Table Ans.61. First 32 Binary and Gray Codes.

```
clear;                              clear;
filename='parrots128'; dim=128;     filename='parrots128'; dim=128;
fid=fopen(filename,'r');            fid=fopen(filename,'r');
img=fread(fid,[dim,dim])';          img=fread(fid,[dim,dim])';
mask=1; % between 1 and 8           mask=1 % between 1 and 8
                                    a=bitshift(img,-1);
                                    b=bitxor(img,a);
nimg=bitget(img,mask);              nimg=bitget(b,mask);
imagesc(nimg), colormap(gray)       imagesc(nimg), colormap(gray)
```

Figure 23.12. Matlab Code to Separate Image Bitplanes.

```
a=linspace(0,31,32); b=bitshift(a,-1);
b=bitxor(a,b); dec2bin(b)
```

Table 23.13. First 32 Binary and Gray Codes.

```
function PSNR(A,B)
if A==B
 error('Images are identical; PSNR is undefined')
end
max2_A=max(max(A)); max2_B=max(max(B));
min2_A=min(min(A)); min2_B=min(min(B));
if max2_A>1 | max2_B>1 | min2_A<0 | min2_B<0
   error('pixels must be in [0,1]')
end
differ=A-B;
decib=20*log10(1/(sqrt(mean(mean(differ.^2)))));
disp(sprintf('PSNR = +%5.2f dB',decib))
```

Figure 23.18. A Matlab Function to Compute PSNR.

```
gamma[i_] := 1. + 2 Floor[Log[2, i]];
Plot[Sum[gamma[j], {j,1,n}]/(n Ceiling[Log[2,n]]), {n,1,200}]
```

Figure 23.30. Gamma Code Versus Binary Code.

```
(* Plot the lengths of four codes
  1. staircase plots of binary representation *)
bin[i_] := 1 + Floor[Log[2, i]];
Table[{Log[10, n], bin[n]}, {n, 1, 1000, 5}];
g1 = ListPlot[%, AxesOrigin -> {0, 0}, PlotJoined -> True,
  PlotStyle -> { AbsoluteDashing[{5, 5}]}]
(* 2. staircase plot of Fibonacci code length *)
fib[i_] := 1 + Floor[ Log[1.618, Sqrt[5] i]];
```

```
Table[{Log[10, n], fib[n]}, {n, 1, 1000, 5}];
g2 = ListPlot[%, AxesOrigin -> {0, 0}, PlotJoined -> True]
(* 3. staircase plot of gamma code length*)
gam[i_] := 1 + 2Floor[Log[2, i]];
Table[{Log[10, n], gam[n]}, {n, 1, 1000, 5}];
g3 = ListPlot[%, AxesOrigin -> {0, 0}, PlotJoined -> True,
  PlotStyle -> { AbsoluteDashing[{2, 2}]}]
(* 4. staircase plot of delta code length*)
del[i_] := 1 + Floor[Log[2, i]] + 2Floor[Log[2, Log[2, i]]];
Table[{Log[10, n], del[n]}, {n, 2, 1000, 5}];
g4 = ListPlot[%, AxesOrigin -> {0, 0}, PlotJoined -> True,
  PlotStyle -> { AbsoluteDashing[{6, 2}]}]
Show[g1, g2, g3, g4, PlotRange -> {{0, 3}, {0, 20}}]
```

Figure 23.31. Lengths of Binary, Fibonacci and Two Elias Codes.

```
i←0; output←null;
repeat
  j←input next chunk;
  (s,i)←Table_i[j];
  append s to output;
until end-of-input
```

Figure 23.41. Fast Huffman Decoding.

Chapter 24

```
p={{5,5},{6, 7},{12.1,13.2},{23,25},{32,29}};
rot={{0.7071,-0.7071},{0.7071,0.7071}};
Sum[p[[i,1]]p[[i,2]], {i,5}]
q=p.rot
Sum[q[[i,1]]q[[i,2]], {i,5}]
```

Figure 24.1. Code for Rotating Five Points.

```
p=Table[Random[Real,{0,2}],{250}];
p=Flatten[Append[p,Table[Random[Real,{1,3}],{250}]]];
p=Flatten[Append[p,Table[Random[Real,{2,4}],{250}]]];
p=Flatten[Append[p,Table[Random[Real,{3,5}],{250}]]];
p=Flatten[Append[p,Table[Random[Real,{4,6}],{250}]]];
p=Flatten[Append[p,Table[Random[Real,{0,6}],{150}]]];
rot={{0.7071,-0.7071},{0.7071,0.7071}};
Graphics[Table[{Hue[RandomReal[]],Point[{p[[i]],p[[i+1]]}]},{i,1,1399,2}],
 Axes->True,AspectRatio->0.5,Ticks->{{{3,128},{6,256}},{{3,128},{6,256}}}]
Graphics[Table[{Hue[RandomReal[]],Point[{p[[i]],p[[i+1]]}.rot]},{i,1,1399,2}],
 Axes->True,AspectRatio->0.5,Ticks->{{{3,128},{6,256}},{{3,128},{-3,-128}}}]
```

Figure 24.2. Rotating a Cloud of Points.

```
filename='lena128'; dim=128;
xdist=zeros(256,1); ydist=zeros(256,1);
fid=fopen(filename,'r');
img=fread(fid,[dim,dim])';
for col=1:2:dim-1
 for row=1:dim
  x=img(row,col)+1; y=img(row,col+1)+1;
  xdist(x)=xdist(x)+1; ydist(y)=ydist(y)+1;
 end
end
figure(1), plot(xdist), colormap(gray) %dist of x&y values
figure(2), plot(ydist), colormap(gray) %before rotation
xdist=zeros(325,1); % clear arrays
ydist=zeros(256,1);
for col=1:2:dim-1
```

```
 for row=1:dim
  x=round((img(row,col)+img(row,col+1))*0.7071);
  y=round((-img(row,col)+img(row,col+1))*0.7071)+101;
  xdist(x)=xdist(x)+1; ydist(y)=ydist(y)+1;
 end
end
figure(3), plot(xdist), colormap(gray) %dist of x&y values
figure(4), plot(ydist), colormap(gray) %after rotation
```

Figure 24.3. Distribution of Image Pixels Before and After Rotation.

```
M=3; N=2^M; H=[1 1; 1 -1]/sqrt(2);
for m=1:(M-1) % recursion
  H=[H H; H -H]/sqrt(2);
end
A=H';
map=[1 5 7 3 4 8 6 2]; % 1:N
for n=1:N, B(:,n)=A(:,map(n)); end;
A=B;
sc=1/(max(abs(A(:))).^2); % scale factor
for row=1:N
  for col=1:N
    BI=A(:,row)*A(:,col).'; % tensor product
    subplot(N,N,(row-1)*N+col)
    oe=round(BI*sc); % results in -1, +1
    imagesc(oe), colormap([1 1 1; .5 .5 .5; 0 0 0])
    drawnow
  end
end
```

Figure Ans.68. The 8×8 WHT Basis Images and Matlab Code.

```
Needs["GraphicsImage`"] (* Draws 2D Haar Coefficients *)
n=8;
h[k_,x_]:=Module[{p,q}, If[k==0, 1/Sqrt[n],           (* h_0(x) *)
 p=0; While[2^p<=k ,p++]; p--; q=k-2^p+1; (* if k>0, calc. p, q *)
 If[(q-1)/(2^p)<=x && x<(q-.5)/(2^p),2^(p/2),
  If[(q-.5)/(2^p)<=x && x<q/(2^p),-2^(p/2),0]]]];
HaarMatrix=Table[h[k,x], {k,0,7}, {x,0,7/n,1/n}] //N;
HaarTensor=Array[Outer[Times, HaarMatrix[[#1]],HaarMatrix[[#2]]]&,
 {n,n}];
Show[GraphicsArray[Map[GraphicsImage[#, {-2,2}]&, HaarTensor,{2}]]]
```

Figure 24.6. The Basis Images of the Haar Transform for $n=8$.

```
n=8;
p={12.,10.,8.,10.,12.,10.,8.,11.};
c=Table[If[t==1, 0.7071, 1], {t,1,n}];
dct[i_]:=Sqrt[2/n]c[[i+1]]Sum[p[[t+1]]Cos[(2t+1)i Pi/16],{t,0,n-1}];
q=Table[dct[i],{i,0,n-1}] (* use exact DCT coefficients *)
q={28,0,0,2,3,-2,0,0}; (* or use quantized DCT coefficients *)
idct[t_]:=Sqrt[2/n]Sum[c[[j+1]]q[[j+1]]Cos[(2t+1)j Pi/16],{j,0,n-1}];
ip=Table[idct[t],{t,0,n-1}]
```

Figure 24.7. Experiments with the One-Dimensional DCT.

```
% 8x8 correlated values
n=8;
p=[00,10,20,30,30,20,10,00; 10,20,30,40,40,30,20,10; 20,30,40,50,50,40,30,20; ...
 30,40,50,60,60,50,40,30; 30,40,50,60,60,50,40,30; 20,30,40,50,50,40,30,20; ...
 10,20,30,40,40,30,12,10; 00,10,20,30,30,20,10,00];
```

```
figure(1), imagesc(p), colormap(gray), axis square, axis off
dct=zeros(n,n);
for j=0:7
 for i=0:7
  for x=0:7
   for y=0:7
dct(i+1,j+1)=dct(i+1,j+1)+p(x+1,y+1)*cos((2*y+1)*j*pi/16)*cos((2*x+1)*i*pi/16);
   end;
  end;
 end;
end;
dct=dct/4; dct(1,:)=dct(1,:)*0.7071; dct(:,1)=dct(:,1)*0.7071;
dct
quant=[239,1,-90,0,0,0,0,0; 0,0,0,0,0,0,0,0; -90,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0; ...
 0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0; 0,0,0,0,0,0,0,0];
idct=zeros(n,n);
for x=0:7
 for y=0:7
  for i=0:7
if i==0 ci=0.7071; else ci=1; end;
   for j=0:7
 if j==0 cj=0.7071; else cj=1; end;
idct(x+1,y+1)=idct(x+1,y+1)+ ...
       ci*cj*quant(i+1,j+1)*cos((2*y+1)*j*pi/16)*cos((2*x+1)*i*pi/16);
   end;
  end;
 end;
end;
idct=idct/4;
idct
figure(2), imagesc(idct), colormap(gray), axis square, axis off
```

Figure 24.19. Code for Highly Correlated Pattern.

```
Table[N[t],{t,Pi/16,15Pi/16,Pi/8}]
dctp[pw_]:=Table[N[Cos[pw t]],{t,Pi/16,15Pi/16,Pi/8}]
dctp[0]
dctp[1]
...
dctp[7]
```

Code for Table 24.23.

```
dct[pw_]:=Plot[Cos[pw t], {t,0,Pi}, DisplayFunction->Identity,
 AspectRatio->Automatic];
dcdot[pw_]:=ListPlot[Table[{t,Cos[pw t]},{t,Pi/16,15Pi/16,Pi/8}],
 DisplayFunction->Identity]
Show[dct[0],dcdot[0], Prolog->AbsolutePointSize[4],
 DisplayFunction->$DisplayFunction]
...
Show[dct[7],dcdot[7], Prolog->AbsolutePointSize[4],
 DisplayFunction->$DisplayFunction]
```

Figure 24.24. A Graphic Representation of the One-Dimensional DCT.

```
dctp[fs_,ft_]:=Table[SetAccuracy[N[(1.-Cos[fs s]Cos[ft t])/2],3],
 {s,Pi/16,15Pi/16,Pi/8},{t,Pi/16,15Pi/16,Pi/8}]//TableForm
dctp[0,0]
dctp[0,1]
...
dctp[7,7]
```

Code for Figure 24.25.

```
Needs["GraphicsImage`"] (* Draws 2D DCT Coefficients *)
```

```
DCTMatrix=Table[If[k==0,Sqrt[1/8],Sqrt[1/4]Cos[Pi(2j+1)k/16]],
 {k,0,7}, {j,0,7}] //N;
DCTTensor=Array[Outer[Times, DCTMatrix[[#1]],DCTMatrix[[#2]]]&,
 {8,8}];
Show[GraphicsArray[Map[GraphicsImage[#, {-.25,.25}]&, DCTTensor,{2}]]]
```

Alternative Code for Figure 24.25.

```
DCTMatrix=Table[If[k==0,Sqrt[1/8],Sqrt[1/4]Cos[Pi(2j+1)k/16]],
 {k,0,7}, {j,0,7}] //N;
DCTTensor=Array[Outer[Times, DCTMatrix[[#1]],DCTMatrix[[#2]]]&,
 {8,8}];
img={{1,0,0,1,1,1,0,1},{1,1,0,0,1,0,1,1},
{0,1,1,0,0,1,0,0},{0,0,0,1,0,0,1,0},
{0,1,0,0,1,0,1,1},{1,1,1,0,0,1,1,0},
{1,1,0,0,1,0,1,1},{0,1,0,1,0,0,1,0}};
ShowImage[Reverse[img]]
dctcoeff=Array[(Plus @@ Flatten[DCTTensor[[#1,#2]] img])&,{8,8}];
dctcoeff=SetAccuracy[dctcoeff,4];
dctcoeff=Chop[dctcoeff,.001];
MatrixForm[dctcoeff]
ShowImage[Reverse[dctcoeff]]
```

Code for Figure 24.26.

```
DCTMatrix=Table[If[k==0,Sqrt[1/8],Sqrt[1/4]Cos[Pi(2j+1)k/16]],
 {k,0,7}, {j,0,7}] //N;
DCTTensor=Array[Outer[Times, DCTMatrix[[#1]],DCTMatrix[[#2]]]&,
 {8,8}];
img={{0,1,0,1,0,1,0,1},{0,1,0,1,0,1,0,1},
 {0,1,0,1,0,1,0,1},{0,1,0,1,0,1,0,1},{0,1,0,1,0,1,0,1},
 {0,1,0,1,0,1,0,1},{0,1,0,1,0,1,0,1},{0,1,0,1,0,1,0,1}};
ShowImage[Reverse[img]]
dctcoeff=Array[(Plus @@ Flatten[DCTTensor[[#1,#2]] img])&,{8,8}];
dctcoeff=SetAccuracy[dctcoeff,4];
dctcoeff=Chop[dctcoeff,.001];
MatrixForm[dctcoeff]
ShowImage[Reverse[dctcoeff]]
```

Code for Figure 24.27.

```
(* DCT-1. Notice (n+1)x(n+1) *)
Clear[n, nor, kj, DCT1, T1];
n=8; nor=Sqrt[2/n];
kj[i_]:=If[i==0 || i==n, 1/Sqrt[2], 1];
DCT1[k_]:=Table[nor kj[j] kj[k] Cos[j k Pi/n], {j,0,n}]
T1=Table[DCT1[k], {k,0,n}]; (* Compute nxn cosines *)
MatrixForm[T1] (* display as a matrix *)
(* multiply rows to show orthonormality *)
MatrixForm[Table[Chop[N[T1[[i]].T1[[j]]]], {i,1,n}, {j,1,n}]]

(* DCT-2 *)
Clear[n, nor, kj, DCT2, T2];
n=8; nor=Sqrt[2/n];
```

```
kj[i_]:=If[i==0 || i==n, 1/Sqrt[2], 1];
DCT2[k_]:=Table[nor kj[k] Cos[(j+1/2)k Pi/n], {j,0,n-1}]
T2=Table[DCT2[k], {k,0,n-1}]; (* Compute nxn cosines *)
MatrixForm[T2] (* display as a matrix *)
(* multiply rows to show orthonormality *)
MatrixForm[Table[Chop[N[T2[[i]].T2[[j]]]], {i,1,n}, {j,1,n}]]

(* DCT-3. This is the transpose of DCT-2 *)
Clear[n, nor, kj, DCT3, T3];
n=8; nor=Sqrt[2/n];
kj[i_]:=If[i==0 || i==n, 1/Sqrt[2], 1];
DCT3[k_]:=Table[nor kj[j] Cos[(k+1/2)j Pi/n], {j,0,n-1}]
T3=Table[DCT3[k], {k,0,n-1}]; (* Compute nxn cosines *)
MatrixForm[T3] (* display as a matrix *)
(* multiply rows to show orthonormality *)
MatrixForm[Table[Chop[N[T3[[i]].T3[[j]]]], {i,1,n}, {j,1,n}]]

(* DCT-4. This is DCT-1 shifted *)
Clear[n, nor, DCT4, T4];
n=8; nor=Sqrt[2/n];
DCT4[k_]:=Table[nor Cos[(k+1/2)(j+1/2) Pi/n], {j,0,n-1}]
T4=Table[DCT4[k], {k,0,n-1}]; (* Compute nxn cosines *)
MatrixForm[T4] (* display as a matrix *)
(* multiply rows to show orthonormality *)
MatrixForm[Table[Chop[N[T4[[i]].T4[[j]]]], {i,1,n}, {j,1,n}]]
```

Figure 24.30. Code for Four DCT Types.

```
function [Q,R]=QRdecompose(A);
% Computes the QR decomposition of matrix A
% R is an upper triangular matrix and Q
% an orthogonal matrix such that A=Q*R.
[m,n]=size(A);  % determine the dimens of A
Q=eye(m);  % Q starts as the mxm identity matrix
R=A;
for p=1:n
 for q=(1+p):m
  w=sqrt(R(p,p)^2+R(q,p)^2);
  s=-R(q,p)/w; c=R(p,p)/w;
  U=eye(m);  % Construct a U matrix for Givens rotation
  U(p,p)=c; U(q,p)=-s; U(p,q)=s; U(q,q)=c;
  R=U'*R;  % one Givens rotation
  Q=Q*U;
 end
end
```

Figure 24.36. A Matlab Function for the QR Decomposition of a Matrix.

```
N=8;
m=[1:N]'*ones(1,N); n=m';
% can also use cos instead of sin
%A=sqrt(2/N)*cos(pi*(2*(n-1)+1).*(m-1)/(2*N));
A=sqrt(2/N)*sin(pi*(2*(n-1)+1).*(m-1)/(2*N));
A(1,:)=sqrt(1/N);
```

```
C=A';
for row=1:N
  for col=1:N
    B=C(:,row)*C(:,col).'; %tensor product
    subplot(N,N,(row-1)*N+col)
    imagesc(B)
    drawnow
  end
end
```

Figure 24.39. The 64 Basis Images of the DST in Two Dimensions.

Chapter 25

```
procedure NWTcalc(a:array of real, n:int);
 comment n is the array size (a power of 2)
 a:=a/√n comment divide entire array
 j:=n;
 while j≥ 2 do
  NWTstep(a, j);
  j:=j/2;
 endwhile;
end;

procedure NWTstep(a:array of real, j:int);
 for i=1 to j/2 do
  b[i]:=(a[2i-1]+a[2i])/√2;
  b[j/2+i]:=(a[2i-1]-a[2i])/√2;
 endfor;
 a:=b; comment move entire array
end;
```

Figure 25.2. Computing the Normalized Wavelet Transform.

```
procedure NWTreconst(a:array of real, n:int);
 j:=2;
 while j≤n do
  NWTRstep(a, j);
  j:=2j;
 endwhile
 a:=a√n; comment multiply entire array
end;

procedure NWTRstep(a:array of real, j:int);
 for i=1 to j/2 do
  b[2i-1]:=(a[i]+a[j/2+i])/√2;
  b[2i]:=(a[i]-a[j/2+i])/√2;
 endfor;
 a:=b; comment move entire array
end;
```

Figure 25.3. Restoring From a Normalized Wavelet Transform.

```
procedure StdCalc(a:array of real, n:int);
```

```
 comment array size is nxn (n = power of 2)
 for r=1 to n do NWTcalc(row r of a, n);
 endfor;
 for c=n to 1 do comment loop backwards
  NWTcalc(col c of a, n);
 endfor;
end;
procedure StdReconst(a:array of real, n:int);
 for c=n to 1 do comment loop backwards
  NWTreconst(col c of a, n);
 endfor;
 for r=1 to n do
  NWTreconst(row r of a, n);
 endfor;
end;
```

Figure 25.6. The Standard Image Wavelet Transform and Decomposition.

```
procedure NStdCalc(a:array of real, n:int);
 a:=a/√n comment divide entire array
 j:=n;
 while j≥ 2 do
  for r=1 to j do NWTstep(row r of a, j);
  endfor;
  for c=j to 1 do comment loop backwards
   NWTstep(col c of a, j);
  endfor;
  j:=j/2;
 endwhile;
end;
procedure NStdReconst(a:array of real, n:int);
 j:=2;
 while j≤n do
  for c=j to 1 do comment loop backwards
   NWTRstep(col c of a, j);
  endfor;
  for r=1 to j do
   NWTRstep(row r of a, j);
  endfor;
  j:=2j;
 endwhile
 a:=a√n; comment multiply entire array
end;
```

Figure 25.7. The Pyramid Image Wavelet Transform.

```
ar=Import["Design.raw", "Bit"];
stp=Partition[ar,256];
{row,col}=Dimensions[stp];
ArrayPlot[stp]
(* step 1, loop over columns and construct array ptp *)
ptp=Table[0,{i,1,row},{j,1,col}];(*Init ptp to zeros*)
mcol=Floor[col/2];
Do[ k=1;
Do[ptp[[i,k]]=(stp[[i,j]]+stp[[i,j+1]])/2;
 ptp[[i,mcol+k]]=(stp[[i,j]]-stp[[i,j+1]])/2; k=k+1,
 {j,1,col-1,2}], {i,1,row}]
```

```
ArrayPlot[ptp]
(* step 2, loop over the rows of ptp and construct array qtp *)
qtp=Table[0,{i,1,row},{j,1,col}];(*Init qtp to zeros*)
mrow=Floor[row/2];
Do[ k=1;
Do[qtp[[k,j]]=(ptp[[i,j]]+ptp[[i+1,j]])/2;
 qtp[[mrow+k,j]]=(ptp[[i,j]]-ptp[[i+1,j]])/2; k=k+1,
 {i,1,row-1,2}], {j,1,col}]
ArrayPlot[qtp]
```

Figure 25.8. A Pyramid Wavelet Decomposition.

```
clear; % main program
filename='lena128'; dim=128;
fid=fopen(filename,'r');
if fid==-1 disp('file not found')
else img=fread(fid,[dim,dim])'; fclose(fid);
end
thresh=0.0;       % percent of transform coefficients deleted
figure(1), imagesc(img), colormap(gray), axis off, axis square
w=harmatt(dim);  % compute the Haar dim x dim transform matrix
timg=w*img*w';   % forward Haar transform
tsort=sort(abs(timg(:)));
tthresh=tsort(floor(max(thresh*dim*dim,1)));
cim=timg.*(abs(timg) > tthresh);
[i,j,s]=find(cim);
dimg=sparse(i,j,s,dim,dim);
% figure(2) displays the remaining transform coefficients
%figure(2), spy(dimg), colormap(gray), axis square
figure(2), image(dimg), colormap(gray), axis square
cimg=full(w'*sparse(dimg)*w);     % inverse Haar transform
density = nnz(dimg);
disp([num2str(100*thresh) '% of smallest coefficients deleted.'])
disp([num2str(density) ' coefficients remain out of ' ...
 num2str(dim) 'x' num2str(dim) '.'])
figure(3), imagesc(cimg), colormap(gray), axis off, axis square

File harmatt.m with two functions

function x = harmatt(dim)
num=log2(dim);
p = sparse(eye(dim)); q = p;
i=1;
while i<=dim/2;
 q(1:2*i,1:2*i) = sparse(individ(2*i));
 p=p*q; i=2*i;
end
x=sparse(p);

function f=individ(n)
x=[1, 1]/sqrt(2);
y=[1,-1]/sqrt(2);
while min(size(x)) < n/2
 x=[x, zeros(min(size(x)),max(size(x)));...
   zeros(min(size(x)),max(size(x))), x];
end
while min(size(y)) < n/2
 y=[y, zeros(min(size(y)),max(size(y)));...
   zeros(min(size(y)),max(size(y))), y];
end
f=[x;y];
```

Figure 25.13. Matlab Code for the Haar Transform of an Image.

```
clear
```

```
a1=[1/2 1/2 0 0 0 0 0 0; 0 0 1/2 1/2 0 0 0 0;
 0 0 0 0 1/2 1/2 0 0; 0 0 0 0 0 0 1/2 1/2;
 1/2 -1/2 0 0 0 0 0 0; 0 0 1/2 -1/2 0 0 0 0;
 0 0 0 0 1/2 -1/2 0 0; 0 0 0 0 0 0 1/2 -1/2];
% a1*[255; 224; 192; 159; 127; 95; 63; 32];
a2=[1/2 1/2 0 0 0 0 0 0; 0 0 1/2 1/2 0 0 0 0;
 1/2 -1/2 0 0 0 0 0 0; 0 0 1/2 -1/2 0 0 0 0;
 0 0 0 0 1 0 0 0; 0 0 0 0 0 1 0 0;
 0 0 0 0 0 0 1 0; 0 0 0 0 0 0 0 1];
a3=[1/2 1/2 0 0 0 0 0 0; 1/2 -1/2 0 0 0 0 0 0;
 0 0 1 0 0 0 0 0; 0 0 0 1 0 0 0 0;
 0 0 0 0 1 0 0 0; 0 0 0 0 0 1 0 0;
 0 0 0 0 0 0 1 0; 0 0 0 0 0 0 0 1];
w=a3*a2*a1;
dim=8; fid=fopen('8x8','r');
img=fread(fid,[dim,dim])'; fclose(fid);
w*img*w' % Result of the transform
```

Figure Ans.72. Code and Results for the Calculation of Matrix W and Transform $W\cdot I\cdot W^T$.

```
function wc1=fwt1(dat,coarse,filter)
%  The 1D Forward Wavelet Transform
%  dat must be a 1D row vector of size 2^n,
%  coarse is the coarsest level of the transform
%  (note that coarse should be <<n)
%  filter is an orthonormal quadrature mirror filter
%  whose length should be <2^(coarse+1)
n=length(dat); j=log2(n); wc1=zeros(1,n);
beta=dat;
for i=j-1:-1:coarse
  alfa=HiPass(beta,filter);
  wc1((2^(i)+1):(2^(i+1)))=alfa;
  beta=LoPass(beta,filter) ;
end
wc1(1:(2^coarse))=beta;

function d=HiPass(dt,filter) % highpass downsampling
d=iconv(mirror(filter),lshift(dt));
% iconv is matlab convolution tool
n=length(d);
d=d(1:2:(n-1));

function d=LoPass(dt,filter) % lowpass downsampling
d=aconv(filter,dt);
% aconv is matlab convolution tool with time-
% reversal of filter
n=length(d);
d=d(1:2:(n-1));

function sgn=mirror(filt)
% return filter coefficients with alternating signs
sgn=-((-1).^(1:length(filt))).*filt;

 A simple test of |fwt1| is

n=16; t=(1:n)./n;
dat=sin(2*pi*t)
filt=[0.4830 0.8365 0.2241 -0.1294];
wc=fwt1(dat,1,filt)

which outputs
```

```
dat=
0.3827  0.7071  0.9239 1.0000 0.9239 0.7071 0.3827 0
-0.3827 -0.7071 -0.9239 -1.0000 -0.9239 -0.7071 -0.3827 0
wc=
1.1365 -1.1365 -1.5685 1.5685 -0.2271 -0.4239 0.2271 0.4239
-0.0281 -0.0818 -0.0876 -0.0421 0.0281 0.0818 0.0876 0.0421
```

Figure 25.21: Code for the One-Dimensional Forward Discrete Wavelet Transform.

```
function dat=iwt1(wc,coarse,filter)
% Inverse Discrete Wavelet Transform
dat=wc(1:2^coarse);
n=length(wc); j=log2(n);
for i=coarse:j-1
 dat=ILoPass(dat,filter)+ ...
  IHiPass(wc((2^(i)+1):(2^(i+1))),filter);
end

function f=ILoPass(dt,filter)
f=iconv(filter,AltrntZro(dt));

function f=IHiPass(dt,filter)
f=aconv(mirror(filter),rshift(AltrntZro(dt)));

function sgn=mirror(filt)
% return filter coefficients with alternating signs
sgn=-((-1).^(1:length(filt))).*filt;

function f=AltrntZro(dt)
% returns a vector of length 2*n with zeros
% placed between consecutive values
n =length(dt)*2; f =zeros(1,n);
f(1:2:(n-1))=dt;
```

A simple test of |iwt1| is

```
n=16; t=(1:n)./n;
dat=sin(2*pi*t)
filt=[0.4830 0.8365 0.2241 -0.1294];
wc=fwt1(dat,1,filt)
rec=iwt1(wc,1,filt)
```

Figure Ans.73: Code for the 1D Inverse Discrete Wavelet Transform.


```
function dat=iwt1(wc,coarse,filter)
% Inverse Discrete Wavelet Transform
dat=wc(1:2^coarse);
n=length(wc); j=log2(n);
for i=coarse:j-1
 dat=ILoPass(dat,filter)+ ...
  IHiPass(wc((2^(i)+1):(2^(i+1))),filter);
end

function f=ILoPass(dt,filter)
f=iconv(filter,AltrntZro(dt));

function f=IHiPass(dt,filter)
f=aconv(mirror(filter),rshift(AltrntZro(dt)));

function sgn=mirror(filt)
% return filter coefficients with alternating signs
sgn=-((-1).^(1:length(filt))).*filt;
```

```
function f=AltrntZro(dt)
% returns a vector of length 2*n with zeros
% placed between consecutive values
n =length(dt)*2; f =zeros(1,n);
f(1:2:(n-1))=dt;

 A simple test of |iwt1| is

n=16; t=(1:n)./n;
dat=sin(2*pi*t)
filt=[0.4830 0.8365 0.2241 -0.1294];
wc=fwt1(dat,1,filt)
rec=iwt1(wc,1,filt)
```

Figure 25.23: Code for the One-Dimensional Inverse Discrete Wavelet Transform.

```
function wc=fwt2(dat,coarse,filter)
%  The 2D Forward Wavelet Transform
%  dat must be a 2D matrix of size (2^n:2^n),
%  "coarse" is the coarsest level of the transform
%  (note that coarse should be <<n)
%  filter is an orthonormal qmf of length<2^(coarse+1)
q=size(dat); n = q(1); j=log2(n);
if q(1)~=q(2), disp('Nonsquare image!'), end;
wc = dat; nc = n;
for i=j-1:-1:coarse,
 top = (nc/2+1):nc; bot = 1:(nc/2);
 for ic=1:nc,
  row = wc(ic,1:nc);
  wc(ic,bot)=LoPass(row,filter);
  wc(ic,top)=HiPass(row,filter);
 end
 for ir=1:nc,
  row = wc(1:nc,ir)';
  wc(top,ir)=HiPass(row,filter)';
  wc(bot,ir)=LoPass(row,filter)';
 end
nc = nc/2;
end

function d=HiPass(dt,filter) % highpass downsampling
d=iconv(mirror(filter),lshift(dt));
% iconv is matlab convolution tool
n=length(d);
d=d(1:2:(n-1));

function d=LoPass(dt,filter) % lowpass downsampling
d=aconv(filter,dt);
% aconv is matlab convolution tool with time-
% reversal of filter
n=length(d);
d=d(1:2:(n-1));

function sgn=mirror(filt)
% return filter coefficients with alternating signs
sgn=-((-1).^(1:length(filt))).*filt;

 A simple test of |fwt2| and |iwt2| is

filename='house128'; dim=128;
fid=fopen(filename,'r');
if fid==-1 disp('file not found')
 else img=fread(fid,[dim,dim])'; fclose(fid);
end
```

```
filt=[0.4830 0.8365 0.2241 -0.1294];
fwim=fwt2(img,4,filt);
figure(1), imagesc(fwim), axis off, axis square
rec=iwt2(fwim,4,filt);
figure(2), imagesc(rec), axis off, axis square
```

Figure 25.24: Code for the Two-Dimensional Forward Discrete Wavelet Transform.

```
function dat=iwt2(wc,coarse,filter)
% Inverse Discrete 2D Wavelet Transform
n=length(wc); j=log2(n);
dat=wc;
nc=2^(coarse+1);
for i=coarse:j-1,
 top=(nc/2+1):nc; bot=1:(nc/2); all=1:nc;
 for ic=1:nc,
  dat(all,ic)=ILoPass(dat(bot,ic)',filter)'  ...
   +IHiPass(dat(top,ic)',filter)';
 end % ic
 for ir=1:nc,
  dat(ir,all)=ILoPass(dat(ir,bot),filter)  ...
   +IHiPass(dat(ir,top),filter);
 end % ir
nc=2*nc;
end % i

function f=ILoPass(dt,filter)
f=iconv(filter,AltrntZro(dt));

function f=IHiPass(dt,filter)
f=aconv(mirror(filter),rshift(AltrntZro(dt)));

function sgn=mirror(filt)
% return filter coefficients with alternating signs
sgn=-((-1).^(1:length(filt))).*filt;

function f=AltrntZro(dt)
% returns a vector of length 2*n with zeros
% placed between consecutive values
n =length(dt)*2; f =zeros(1,n);
f(1:2:(n-1))=dt;

A simple test of |fwt2| and |iwt2| is

filename='house128'; dim=128;
fid=fopen(filename,'r');
if fid==-1 disp('file not found')
 else img=fread(fid,[dim,dim])'; fclose(fid);
end
filt=[0.4830 0.8365 0.2241 -0.1294];
fwim=fwt2(img,4,filt);
figure(1), imagesc(fwim), axis off, axis square
rec=iwt2(fwim,4,filt);
figure(2), imagesc(rec), axis off, axis square
```

Figure 25.25: Code for the Two-Dimensional Inverse Discrete Wavelet Transform.

```
clear, colormap(gray);
filename='lena128'; dim=128;
fid=fopen(filename,'r');
img=fread(fid,[dim,dim])';
filt=[0.23037,0.71484,0.63088,-0.02798, ...
 -0.18703,0.03084,0.03288,-0.01059];
fwim=fwt2(img,3,filt);
```

```
figure(1), imagesc(fwim), axis square
 fwim(1:16,17:32)=fwim(1:16,17:32)/2;
 fwim(1:16,33:128)=0;
 fwim(17:32,1:32)=fwim(17:32,1:32)/2;
 fwim(17:32,33:128)=0;
 fwim(33:128,:)=0;
figure(2), colormap(gray), imagesc(fwim)
rec=iwt2(fwim,3,filt);
figure(3), colormap(gray), imagesc(rec)
```

Code for Figure Ans.74.

1. Initialization:
    Initialize LP with all $C_{i,j}$ in LFS,
    Initialize LIS with all parent nodes,
    Output $n = \lfloor \log_2(\max|C_{i,j}|/q) \rfloor$.
    Set the threshold $T = q2^n$, where q is a quality factor.
2. Sorting:
    for each node k in LIS do
      output $S_T(k)$
      if $S_T(k) = 1$ then
        for each child of k do
          move coefficients to LP
          add to LIS as a new node
        endfor
        remove k from LIS
      endif
    endfor
3. Quantization: For each element in LP,
    quantize and encode using ACTCQ.
    (use TCQ step size $\Delta = \alpha \cdot q$).
4. Update: Remove all elements in LP. Set $T = T/2$. Go to step 2.

Figure 25.35. QTCQ Encoding.

Chapter 26

```
gamma = 2.2;
Show[Graphics[
 Table[{GrayLevel[x], Rectangle[{x,0},{x+.01,0.1}]},{x,0,1,0.01}]],
 Graphics[{GrayLevel[0], Rectangle[{1,0},{1.001,0.1}]}]]
Show[Graphics[
 Table[{GrayLevel[x^gamma], Rectangle[{x,0},{x+.01,0.1}]},{x,0,1,0.01}]],
 Graphics[{GrayLevel[0], Rectangle[{1,0},{1.001,0.1}]}]]
```

Figure 26.10. The Gamma Transform.

```
gamma = 0.45;
Plot[x^gamma, {x, 0, 1}]
```

Figure 26.11. NTSC Gamma Correction Curve.

Appendix D

```
(* non-barycentric weights example *)
Clear[p0,p1,g1,g2,g3,g4];
p0={0,0}; p1={5,6};
g1=ParametricPlot[(1-t)^3 p0+t^3 p1,{t,0,1}, PlotRange->All, Compiled->False,
 DisplayFunction->Identity];
g3=Graphics[{AbsolutePointSize[4],{Point[p0],Point[p1]} }];
```

```
p0={0,-1}; p1={5,5};
g2=ParametricPlot[(1-t)^3 p0+t^3 p1,{t,0,1},PlotRange->All, Compiled->False,
 PlotStyle->AbsoluteDashing[{2,2}], DisplayFunction->Identity];
g4=Graphics[{AbsolutePointSize[4], {Point[p0],Point[p1]} }];
Show[g2,g1,g3,g4, DisplayFunction->$DisplayFunction, DefaultFont->{"cmr10", 10}];
```

Non-barycentric weights example.

```
(* a bilinear surface patch *)
Clear[bilinear,pnts,u,w];
<<:Graphics:ParametricPlot3D.m;
pnts=ReadList["Points",{Number,Number,Number}, RecordLists->True];
bilinear[u_,w_]:=pnts[[1,1]](1-u)(1-w)+pnts[[1,2]]u(1-w)\
 +pnts[[2,1]]w(1-u)+pnts[[2,2]]u w;
Simplify[bilinear[u,w]]
g1=Graphics3D[{AbsolutePointSize[5], Table[Point[pnts[[i,j]]],{i,1,2},{j,1,2}]}];
g2=ParametricPlot3D[bilinear[u,w],{u,0,1,.05},{w,0,1,.05}, Compiled->False,
 DisplayFunction->Identity];
Show[g1,g2, ViewPoint->{0.063, -1.734, 2.905}];
```

A bilinear Surface Patch.

```
(* A Rational Bezier Surface *)
Clear[pwr,bern,spnts,n,m,wt,bzSurf,cpnts,patch,vlines,hlines,axes];
<<:Graphics:ParametricPlot3D.m
spnts={{{0,0,0},{1,0,1},{0,0,2}},
 {{1,1,0},{4,1,1},{1,1,2}}, {{0,2,0},{1,2,1},{0,2,2}}};
m=Length[spnts[[1]]]-1; n=Length[Transpose[spnts][[1]]]-1;
wt=Table[1, {i,1,n+1},{j,1,m+1}];
wt[[2,2]]=5;
pwr[x_,y_]:=If[x==0 && y==0, 1, x^y];
bern[n_,i_,u_]:=Binomial[n,i]pwr[u,i]pwr[1-u,n-i]
bzSurf[u_,w_]:=
 Sum[wt[[i+1,j+1]]spnts[[i+1,j+1]]bern[n,i,u]bern[m,j,w], {i,0,n}, {j,0,m}]/
 Sum[wt[[i+1,j+1]]bern[n,i,u]bern[m,j,w], {i,0,n}, {j,0,m}];
patch=ParametricPlot3D[bzSurf[u,w],{u,0,1}, {w,0,1},
 Compiled->False, DisplayFunction->Identity];
cpnts=Graphics3D[{AbsolutePointSize[4], (* control points *)
 Table[Point[spnts[[i,j]]], {i,1,n+1},{j,1,m+1}]}];
vlines=Graphics3D[{AbsoluteThickness[1], (* control polygon *)
 Table[Line[{spnts[[i,j]],spnts[[i+1,j]]}], {i,1,n}, {j,1,m+1}]}];
hlines=Graphics3D[{AbsoluteThickness[1],
 Table[Line[{spnts[[i,j]],spnts[[i,j+1]]}], {i,1,n+1}, {j,1,m}]}];
maxx=Max[Table[Part[spnts[[i,j]], 1], {i,1,n+1}, {j,1,m+1}]];
maxy=Max[Table[Part[spnts[[i,j]], 2], {i,1,n+1}, {j,1,m+1}]];
maxz=Max[Table[Part[spnts[[i,j]], 3], {i,1,n+1}, {j,1,m+1}]];
axes=Graphics3D[{AbsoluteThickness[1.5], (* the coordinate axes *)
 Line[{{0,0,maxz},{0,0,0},{maxx,0,0},{0,0,0},{0,maxy,0}}]}];
Show[cpnts,hlines,vlines,axes,patch, PlotRange->All, DefaultFont->{"cmr10",10},
 DisplayFunction->$DisplayFunction, ViewPoint->{2.783, -3.090, 1.243}];
```

Code for Figure 13.42 (a rational Bézier surface patch).

```
pnts={{{0,1,0},{1,1,1},{2,1,0}},{{0,0,0},{1,0,0},{2,0,0}}};
b1[w_]:={1-w,w}; b2[u_]:={(1-u)^2,2u(1-u),u^2};
comb[i_]:=(b1[w].pnts)[[i]] b2[u][[i]];
g1=ParametricPlot3D[comb[1]+comb[2]+comb[3], {u,0,1},{w,0,1}, Compiled->False,
DefaultFont->{"cmr10", 10}, DisplayFunction->Identity,
 AspectRatio->Automatic, Ticks->{{0,1,2},{0,1},{0,.5}}];
```

Code for Figure 13.34 (a lofted Bézier surface patch).

```
m={{m11,m12,m13},{m21,m22,m23}}; a={a1,a2}; b={b1,b2,b3};
a.m.b
```

Test of the above code.

```
(* Degree elevation of a rect Bezier surface from 2x3 to 4x5 *)
Clear[a,p,q,r];
```

```
m=1; n=2;
p={{p00,p01,p02},{p10,p11,p12}}; (* array of points *)
r=Array[a, {m+3,n+3}]; (* extended array, still undefined *)
Part[r,1]=Table[a, {i,-1,m+2}];
Part[r,2]=Append[Prepend[Part[p,1],a],a];
Part[r,3]=Append[Prepend[Part[p,2],a],a];
Part[r,n+2]=Table[a, {i,-1,m+2}];
MatrixForm[r] (* display extended array *)
q[i_,j_]:=({i/(m+1),1-i/(m+1)}. (* dot product *)
 {{r[[i+1,j+1]],r[[i+1,j+2]]},{r[[i+2,j+1]],r[[i+2,j+2]]}}).
 {j/(n+1),1-j/(n+1)}
q[2,3] (* test *)
```

Figure 13.37 (code for degree elevation of a rectangular Bézier surface).
[End of listings for the Manual of Computer Graphics, April 2011.]