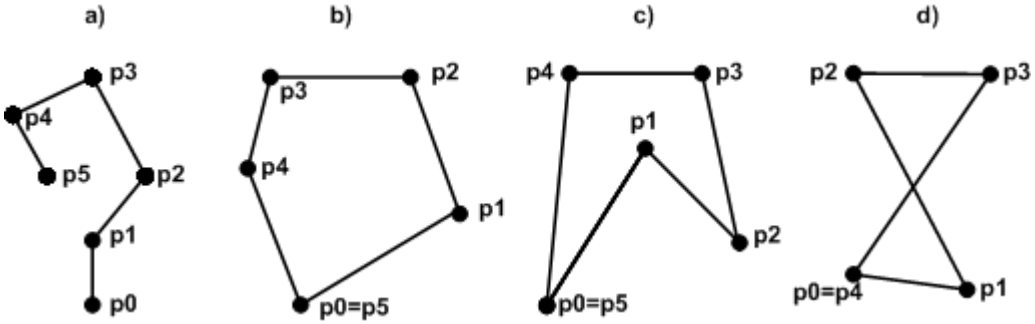
Fragen und Antworten: 2D-Vektorgraphik

2D Vektorgraphik

F: **Je ein Beispiel mit Vertexnummern für**
a) offenes Polygon = polyline
b) konvexes Polygon
c) konkaves Polygon
d) überschlagenes Polygon = nonsimple polygon
A:



F: **Speicherbedarf eines 2D-Polygons mit N Ecken und float-Koordinaten ?**
A: Speicherplatz in Bytes = `N * 2 * SizeOf( float )`

F: **Die xy-Koordinaten der Polygone sind fast immer vom Typ FLOAT oder DOUBLE.**
In welchen Fällen sind sie gegen die Regel vom Typ INTEGER ?
A: bei Bildschirmkoordinaten, bei Koordinaten, die von OnMouseDown, OnMouseMove etc. geliefert werden, bei der Polygonisierung von Rastergraphiken

F: **Ein Polygon sei in einem Array `p[n]` vom Typ `Point` bzw. `CPoint` gespeichert.**
Schreiben Sie ein Mini-Programm, welches das Polygon zeichnet in C# oder C++ oder Java.
A: in C#:
```
Graphics g = CreateGraphics(); Pen mypen = new Pen( Color.Black,1 );
for ( int i = 0; i < n-1; i++ ) g.DrawLine( mypen, p[i], p[i+1] );
```
in C++:
```
CClientDC dc(this);
dc.MoveTo( p[0] ); for ( int i = 1; i < n; i++ ) dc.LineTo( p[i] );
```
in Java:
```
Graphics g = getGraphics();
for ( int i = 0; i < n-1; i++ ) g.drawLine( p[i].x, p[i].y, p[i+1].x, p[i+1].y );
```

F: **Erklärung der 2D-Graphikbefehle: `(Draw)PolyLine`, `(Draw)Polygon` und `(Draw)PolyBezier` ?**

	`PolyLine`	`Polygon`	`PolyBézier`
automatisches schließen:			
Fläche füllen:			
Anzahl der Punkte:			
Verbindungen:			
Eckige Vertices:			
Biegungen:			

A:

	`PolyLine`	`Polygon`	`PolyBézier`
automatisches schließen:	nein	ja	nein
Fläche füllen:	nein	ja	nein
Anzahl der Punkte:	`n` beliebig	`n` beliebig	`n = 4,7,10,13,` etc.
Verbindungen:	`n-1` Strecken von `p[i]` nach `p[i+1]`	`n-1` Strecken von `p[i]` nach `p[i+1]`	`n/3` Polynome 3. Grades von `p[i]` nach `p[i+3]`
Eckige Vertices:	alle `n`	alle `n`	`i = 0,3,6,9,12,` etc.
Biegungen:	keine	keine	zu allen Vertices, die nicht eckig sind

Alle drei Befehle erwarten als Parameter ein Polygon = Array mit Elementen vom Typ `Point` bzw. `CPoint`.
`PolyLine` zeichnet nur Linien, keine Flächen und lässt ein offenes Polygon offen,
`Polygon` schließt ein offenes Polygon und füllt dessen Fläche mit Farbe,
`PolyBezier` erwartet ein Polygon der Länge 4, 7, 10, 13 etc. Es berechnet für jeweils 4 Punkte eine Bezierkurve, wobei der erste und letzte exakt getroffen werden, die beiden mittleren nicht, sondern nur die Biegung beeinflussen, ansonsten wie `PolyLine`.

F: **Programm, das den Eckenschwerpunkt `xs`, `ys` eines geschlossenen Polygons `p[n]` berechnet ?**
A:
```
float xs = 0, ys = 0;
for ( int i = 0; i < n-1; i++ ){ xs += p[i].X; ys += p[i].Y; }
xs /= n-1; ys /= n-1;
```
F: **Programm, das den Umfang eines Polygons `p[n]` berechnet ?**
A:
```
double umfang = 0;
for ( int i = 0; i < n-1; i++ )
{ double dx = p[i+1].X - p[i].X;
  double dy = p[i+1].Y - p[i].Y;
  umfang += sqrt( dx*dx + dy*dy );
}
```
F: **Programm, das die Fläche eines Polygons `p[n]` berechnet ?**
A:
```
float flaeche = 0;
for ( int i = 0; i < n-1; i++ )
{ float dx =  p[i+1].X - p[i].X;
  float my = (p[i+1].Y + p[i].Y) / 2f;
  flaeche += dx * my;
}
```
F: **Programm, das das umschreibende Rechteck `xmin,ymin,xmax,ymax` eines Polygons `p[n]` berechnet ?**
A:
```
float xmin = p[0].X, ymin = p[0].Y, xmax = p[0].X, ymax = p[0].Y;
for ( int i = 1; i < n; i++ )
{ if ( p[i].X < xmin ) xmin = p[i].X;
  if ( p[i].X > xmax ) xmax = p[i].X;
  if ( p[i].Y < ymin ) ymin = p[i].Y;
  if ( p[i].Y > ymax ) ymax = p[i].Y;
}
```

F: **Programm, das ein Polygon `p[n]` um `zx` und `zy` zoomt (Streckungs-/Stauchungszentrum `xm`,`ym`) ?**
A:
```
for ( int i = 0; i < n; i++ )
{ p[i].X = ( p[i].X - xm ) * zx + xm;
  p[i].Y = ( p[i].Y - ym ) * zy + ym;
}
```
F: **Programm, das ein Polygon `p[n]` um den Winkel 17 Grad dreht (Drehpunkt `xm`,`ym`) ?**
A:
```
double arcus = (2 * Math.PI * 17) / 360;
float cosinus = (float)Math.Cos( arcus );
float sinus   = (float)Math.Sin( arcus );
for ( int i = 0; i < n; i++ )
{ float x = p[i].X - xm;
  float y = p[i].Y - ym;
  p[i].X = cosinus * x - sinus * y + xm;
  p[i].Y = sinus * x + cosinus * y + ym;
}
```
F: **Definition eines beliebigen Brush und eines beliebigen Pen (C# oder C++/MFC oder Java) ?**
A: C#: `Brush mybrush = new SolidBrush(Color.Red); Pen mypen = new Pen(Color.Red,5);`
C++/MFC: `CPen mypen; CBrush mybrush; mypen.CreatePen(PS_SOLID, 5, RGB( 255,0,0 ) );`
`mybrush.CreateSolidBrush( RGB( 255,0,0 ) );`
Java: `g.SetColor(Color.red); BasicStroke mypen = new BasicStroke( 5 );`
Einen Brush kennt Java nicht.

F: **Eine `OnPaint`-Funktion, die beide Diagonalen in ihr Fenster schreibt. (C# oder Java oder C++)**
```
protected override void OnPaint( PaintEventArgs e )
{ Graphics g = e.Graphics;
  ....................................
  Pen blackpen = new Pen( Color.Black, 4 );
  g.DrawLine( ..................................... );
  g.DrawLine( ..................................... );
}
```
A:
```
protected override void OnPaint( PaintEventArgs e )
{ Graphics g = e.Graphics;
  Rectangle cr = ClientRectangle;
  Pen blackpen = new Pen( Color.Black, 4 );
  g.DrawLine( blackpen, 0, 0, cr.Width, cr.Height );
  g.DrawLine( blackpen, cr.Width, 0, 0, cr.Height );
}
```

F: **Eine `OnPaint`-Funktion, die eine gefüllte Ellipse (schwarzer Rand der Dicke 4, innen Zufallsfarbe) zentriert (Randabstand je 20%) in ihr Fenster schreibt (C# oder Java oder C++).**
```
protected override void OnPaint( PaintEventArgs e )
{ Graphics g = e.Graphics;
  ....................................
  Pen blackpen = new Pen( Color.Black, 4 );
  Random r = new Random();
  Byte red   = ..............................
  Byte green = ..............................
  Byte blue  = ..............................
  Brush brush = new SolidBrush( ................................... );
  g.FillEllipse( brush,.................................................... );
  g.DrawEllipse( blackpen,................................................. );
}
```
A:
```
protected override void OnPaint( PaintEventArgs e )
{ Graphics g = e.Graphics;
  Rectangle cr = ClientRectangle;
  Pen blackpen = new Pen( Color.Black, 4 );
  Random r = new Random();
  Byte red   = (Byte)r.Next( Byte.MaxValue );
  Byte green = (Byte)r.Next( Byte.MaxValue );
  Byte blue  = (Byte)r.Next( Byte.MaxValue );
  Brush brush = new SolidBrush( Color.FromArgb( red, green, blue ) );
  g.FillEllipse( brush   , cr.Width/5, cr.Height/5, (3*cr.Width)/5, (3*cr.Height)/5 );
  g.DrawEllipse( blackpen, cr.Width/5, cr.Height/5, (3*cr.Width)/5, (3*cr.Height)/5 );
}
```

F: **Eine `OnPaint`-Funktion, die aus der Mitte des Fensters einen Stern von 360 Linien von zufälliger Länge mit zufälligen Farben zeichnet. Ergänzen Sie die Stellen mit den Punkten.**

```
protected override void OnPaint( PaintEventArgs e )
{ Double radius_x, radius_y, arcus_1, arcus_i, factor, sinus, cosinus;
  Rectangle cr = ClientRectangle;
  radius_x = .................;//halbe Breite
  radius_y = .................;//halbe Höhe
  int mid_x = (int)radius_x;
  int mid_y = (int)radius_y;
  arcus_1 = .................;//ein Grad in Bogenmaß
  Random r = new Random();
  for ( int i ............... )//pro Grad eine Linie
  { Pen mypen  = new Pen( Color.FromArgb(r.Next( 255 ),r.Next( 255 ),r.Next( 255 ) ), 1 );
    factor = ........................................//Zufallsfaktor zwischen 0.25 und 1
    arcus_i = ........................................//aktueller Winkel
    cosinus = ........................................//Ankathete
    sinus   = ........................................//Gegenkathete
    e.Graphics.DrawLine( mypen, ..................................................);
    mypen.Dispose();
  }
}
A:
protected override void OnPaint( PaintEventArgs e )
{ Double radius_x, radius_y, arcus_1, arcus_i, factor, sinus, cosinus;
  Rectangle cr = ClientRectangle;
  radius_x = cr.Width  / 2;
  radius_y = cr.Height / 2;
  int mid_x = (int)radius_x;
  int mid_y = (int)radius_y;
  arcus_1 = 2.0 * Math.PI / 360.0;
  Random r = new Random();
  for ( int i=0; i < 360; i++ )
  { Pen mypen  = new Pen( Color.FromArgb(r.Next( 255 ),r.Next( 255 ),r.Next( 255 ) ), 1 );
    factor = Math.Max( 0.25, r.NextDouble() );
    arcus_i = arcus_1 * i;
    cosinus = radius_x * factor * Math.Cos( arcus_i );
    sinus   = radius_y * factor * Math.Sin( arcus_i );
    e.Graphics.DrawLine( mypen, mid_x, mid_y, mid_x + (int)cosinus, mid_y + (int)sinus );
    mypen.Dispose();
  }
}
```

F: **Minimales Malprogramm in C# oder Pseudocode mit folgenden globalen Variablen:**

```
int x0, y0;
Graphics g = CreateGraphics();
Pen mypen = new Pen( Color.Red, 5 );
```

A:

```
protected override void OnMouseDown( MouseEventArgs e )
{ x0 = e.X; y0 = e.Y;
}
protected override void OnMouseMove( MouseEventArgs e )
{ if ( e.Button == MouseButtons.None ) return;
  g.DrawLine( mypen, x0, y0, e.X, e.Y );
  x0 = e.X; y0 = e.Y;
}
```

F: **Was ist XAML und SVG ?**

	XAML	SVG
Abkürzung von:		
Eigentümer:		
enthalten in:		
nachrüsten durch PlugIn:		
Zukunft:		

A:

	XAML	SVG
Abkürzung von:	Extensible Application Markup Language	Scalable Vector Graphics
Eigentümer:	Microsoft	Adobe
nativ enthalten in:	Internet Explorer ab Vers. 7.0	Mozilla Firefox ab Vers. 2.0
nachrüsten durch PlugIn:	für alle Browser bei Microsoft	für alle Browser bei Adobe
Zukunft:	wird Standard	verschwindet

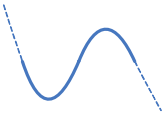
Wie Flash (Macromedia) sind SVG und XBAP (=XAML Browser Applications) geeignet für den Transport und die Darstellung von Vektorgraphiken im WEB. Anders als Flash basieren beide auf XML. XAML ist mächtiger als SVG (3D und User Interfaces für Windows Presentation Foundation WPF).

F: **a) Was ist ein Polygon ? b) Was ist ein Polynom ? c) Was haben Polynome mit Polygonen zu tun ?**
A kurz:
a) geordnete Menge von Vertices (`xi,yi`)
b) Funktion vom Typ Gerade oder Parabel oder 2-bogig oder 3-bogig usw.
c) Ein Polynom approximiert oder interpoliert ein Polygon.
A lang:
a) Polygon = geordnete Menge von Eckpunkten = Vertices (`x0,y0), (x1,y1),...(xi,yi)...(xn,yn)`. Wenn `xn=x0` und `y0=yn`, dann ist das Polygon geschlossen (umschließt eine Gebiet), ansonsten ist es offen.
b) Polynom = Funktion vom Typ: $y = a_{n-1}*x^{n-1} + a_{n-2}*x^{n-2} + a_{n-3}*x^{n-3} + ... + a_2*x^2 + a_1*x + a_0$ einfache Beispiele:

Gerade =
Polynom 1. Grades
=
$y = a*x + b$
hat keine Biegung
wird durch 2
Punkte definiert

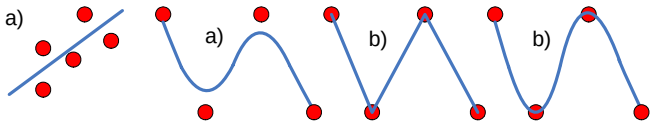
Parabel =
Polynom 2. Grades
=
$y = a*x^2 + b*x + c$
hat eine Biegung
wird durch 3
Punkte definiert

kubisches Polynom =
Polynom 3. Grades =
$y = a*x^3 + b*x^2 + c*x + d$
hat zwei Biegungen
wird durch 4 Punkte
definiert



c) Man kann Polygone durch Polynome approximieren und interpolieren.

F: **Was ist a) Approximation und b) Interpolation ? Je zwei Beispiele zeichnen.**
A kurz:
a) Annäherung: Die Kurve hat eine gewisse Steifigkeit. Die Kurve trifft die Punkte in der Regel nicht.
b) Zwischenpolbildung: Die Kurve besitzt keine Steifigkeit. Sie verbindet immer alle Punkte.



A lang: Approximation = wörtlich übersetzt: Annäherung = Finden einer Funktion so, dass sie trotz ihrer Steifigkeit einer vorgegebene Punktmenge = Stützpunkte oder Knotenpunkte möglichst gut folgt.
Beispiel1: Bestimme eine Gerade durch eine Punktwolke so, dass die Summe der Abstände minimal ist.
Beispiel2: Bestimme die Koeffizienten `a3, a2, a1, a0` des Polynoms
$y = a3*x^3 + a2*x^2 + a1*x + a0$ mit Hilfe von 4 Punkten so, dass der erste und der letzte getroffen werden und dass die beiden mittleren die zwei Biegungen bestimmen.
Interpolation = wörtlich übersetzt: Zwischenpolbildung = Finden von verbindenden Zwischenpunkten zwischen zwei oder mehr auseinander liegenden Eckpunkten. Einfachstes Beispiel für lineare Interpolation = zeichnen einer Rasterstrecke = `g.DrawLine( mypen, x0, y0, x1, y1 );`.

F: **Was versteht man unter einer Bézier-Kurve ? Hauptanwendung ?**
A kurz: Approximation eines n-eckigen Polygons durch ein Polynom [n-1]ten Grades. Hauptanw. = Approx. von je 4 aufeinander folgenden Punkten durch je ein kubisches Polynom bei Schriftumrissen: True Type Fonts = TTF und PostScript Fonts.
A lang: Von Pierre Bézier und Paul de Casteljau1962 erfundene Approximation eines n-Ecken-Polygons durch ein Polynom [n-1]ten Grades. Polygon und Polynom beginnen und enden am gleichen Punkt, die Zwischenpunkte des Polygons werden nicht getroffen, sondern bestimmen nur die lokalen Biegungen. Wichtigster Sonderfall = kubische Bézier-Kurven = Polynome 3. Grades = Typ: $y = a3*x^3 + a2*x^2 + a1*x + a0$ = Randkurven aller stufenlos zoombaren True-Type-Vektorschriften. Sie verbinden jeweils die erste und letzte von 4 aufeinander folgenden Polygonecken. Sind die 4 Punkte kolinear, dann entsteht eine Gerade, wenn nicht, biegen die beiden mittleren Polygonecken eine Kurve.

F: **Was ist ein Spline ?**
A kurz: Planke. Interpolation. Zusammensetzen von Stücken von $y = a3*x^3 + a2*x^2 + a1*x + a0$ so, dass an den Stoßstellen kein Knick zu sehen ist.
A lang: Spline = wörtlich übersetzt: Planke = Interpolation von Polygonen durch zwei oder mehr hintereinander folgende Polynome so, dass die Illusion einer einzigen, geschlossenen Kurve entsteht. Um diese Illusion zu erreichen, müssen die Ableitungen am Endpunkt jedes Polynoms i identisch sein mit denen am Anfang des Folgepolynoms i+1. Häufigste Splines = kubische Splines = Polynome vom Typ: $y = a3*x^3 + a2*x^2 + a1*x + a0.$ Anders als Bézier-Kurven treffen Splines alle Polygonecken exakt.

F: **Was ist ein NURBS ?**
A kurz: Non Uniform Rational B-Spline. Allgemeinster 3D-Kurventyp zur Approx. oder Interpol. aller Arten von Polygonen.
A lang: Non Uniform bedeutet, dass die Polygoneckenabstände nicht gleich sein müssen (keine Äquidistanz notwendig).
Rational bedeutet, dass die Basisfunktionen nicht Polynome sind, sondern deren Quotienten: Polynom dividiert durch Polynom.
Verallgemeinert 3D-Flächen, Geraden, Bézier-Kurven, Splines und alle Mischformen.

F: **Gerade durch x_0, y_0 and x_1, y_1 in Parameterform ?**
A: $x = x_0 + t * (x_1 - x_0)$
$y = y_0 + t * (y_1 - y_0)$

F: **Programm, das 100 gleichverteilte Punkte zwischen x0,y0 und x1,y1 interpoliert ?**
A:
```
float[] x = new Single[100]; float[] y = new Single[100];
for ( int i=0; i < 100; i++ )
{ float t = i * 0.01f;
  x[i] = x0 + t * ( x1 - x0 );
  y[i] = y0 + t * ( y1 - y0 );
}
```

F: **Programm, das 100 gleichverteilte Punkte auf einem Kreis mit Radius r und Mittelpunkt xm,ym berechnet ?**
A:
```
float[] x = new Single[100]; float[] y = new Single[100];
for ( int i=0; i < 100; i++ )
{ double arcus = 2.0 * Math.PI * i * 0.01;
  x[i] = xm + r * (Single)Math.Cos( arcus );
  y[i] = ym + r * (Single)Math.Sin( arcus );
}
```