Fragen und Antworten: Rastergraphik

F: **Aufbau der Hauptdatenstrukturen von a) 2D-Vektorgraphik und b) 2D-Rastergraphik ?**
A kurz: a) Polygon = $p(x_0,y_0), p(x_1,y_1), \ldots, p(x_i,y_i), \ldots p(x_{n-1},y_{n-1})$.
b) Matrix = `bild[y,x]` mit Zeilenindex `0 <= y <= ySize-1` und Spaltenindex `0 <= x <= xSize-1`.
A lang: a) Polygon = geordnete Menge von Punkten `p[0], p[1], ... p[i] ... p[n-1]`,
wobei jeder `p[i]` ein `float x` und ein `float y` enthält. Keine Strecke `p[i],p[i+1]` darf sich mit einer anderen Strecke `p[j],p[j+1]` überkreuzen. Wenn `p[0]` identisch mit `p[n-1]`,
dann ist das Polygon geschlossen, andernfalls ist es offen.
b) Matrix = rechteckige Anordnung von Zahlen in `xSize` Spalten und `ySize` Zeilen.
Spaltenindex: `0 <= x <= xSize-1`, Zeilenindex: `0 <= y <= ySize-1`

F: **Definitionen von Diskretisierung, Quantisierung, Digitalisierung, Binarisierung ?**
A: Diskr. = Abbildung der realen Welt auf eine rechteckige Sensormatrix (meistens CCD)
Quant.= Abbildung der analogen Helligkeitsachse auf eine geordnete Menge ganzer Zahlen (meistens 0 - 255)
Dig. = Hintereinander Ausführung von Diskr. und Quant.
Bin. = Quant. mit genau 2 Grauwerten (meistens 0 und 1 oder 0 und 255)

F: **Das erste Pixel eines Bildes habe die Adresse: `Byte* p;`. Welche lineare Adresse hat ein Pixel in der Zeile `y` und der Spalte `x` ? Erklären Sie die Formel anschaulich !**
A: `Adresse = p + y * xSize + x`
Schubladenanalogie: `y` Schubladen der Breite `xSize` und werden hinter `p` nebeneinander linear aufgereiht.
`x` ist der Versatz in der letzten Schublade.

F: **Erklären Sie folgende Pixel-Formate**:

Format	Bildtyp	mögliche Farben	Anwendung
1bppIndexed			
4bppIndexed			
8bppIndexed			
16bppGrayScale			
16bppRgb555			
24bppRgb			
32bppArgb			
64bppArgb			

A:

Format	Bildtyp	mögliche Farben	Anwendung
1bppIndexed	Falschfarbe	2	Binärbilder, s/w-Drucker
4bppIndexed	Falschfarbe	2^4 = 16	Icons
8bppIndexed	Falschfarbe	2^8 = 256	hand held games
16bppGrayScale	Grauwerte	2^{16} Graustufen	Medizin
16bppRgb555	True Color	2^{16} = 65.536	low quality photos
24bppRgb	True Color	2^{24} = 16.777.216	Digitalcameras
32bppArgb	True Color	2^{24} + 256 Transparenzstufen	Virtual Reality
64bppArgb	True Color	2^{48} + 2^{16} Transparenzstufen	Satellitenbilder

F: **Erklären Sie die Vorgänge, wenn ein Rasterbild a) um 10 % vergrößert und b) um 10% verkleinert wird und wenn es c) in zwei Schritten um je 10% vergrößert wird und d) einmal um 20% vergrößert wird.**
A: a) Es werden jede zehnte Spalte und jede zehnte Zeile verdoppelt ohne Rücksicht auf die Bildverzerrung.
b) Es werden jede zehnte Spalte und Zeile weggelassen und alle Nachfolgespalten und Zeilen werden auf die freien Plätze vorgezogen ohne Rücksicht auf Bildverlust und Bildverzerrung.
c) Die zweifache Verdoppelung führt zu starken Verzerrungen.
Beispiel: 0123456789 → 01234567899 → 012345678999
d) Jede fünfte Spalte und Zeile wird verdoppelt. Beispiel: 0123456789 → 012344567899

F: **Fähigkeiten von Vektorgraphik und Rastergraphik**

	Vektorgraphik	Rastergraphik
Linie zeichnen		
Fläche füllen		
Schrift		
Bilder der realen Welt		

A:

	Vektorgraphik	Rastergraphik
Linie zeichnen	ideal, hochpräzise	nur horizontal und vertikal, alle schrägen Linien nur als Treppen
Fläche füllen	keine Flächenfüllung möglich, nur Schraffur	gut, jedoch mit treppigen Rändern
Schrift	zu dünn, aber ideal für skalierbare Umrisse (True Type Font)	gut, aber jeder Rasterfont braucht für jede Größe je einen Font
Bilder der realen Welt	nur in Umrissen und Schraffuren wie bei Kupferstichen	gut, siehe TV

F: **Ein Polygon `p0` wird nach `p1` transformiert und ein Rasterbild `bmp0` nach `bmp1`.**
Was sind die wichtigsten Unterschiede der beiden Transformationen ?

	Polygon Transf. = Vektor Operationen scroll, zoom, rot	Bitmap Transf. = Raster Operationen scroll, zoom, rot
Typ von `x,y`		
Stufen		
Präzision		
Richtung		
Rand		
rückgängig		
stapelbar		
überschreiben		

A:

	Polygon Transf. = Vektor Operationen scroll, zoom, rot	Bitmap Transf. = Raster Operationen scroll, zoom, rot
Typ von `x,y`	alle `x,y` sind reelle Zahlen (meist `float`)	alle `x,y` sind ganze Zahlen (meist `int`)
Stufen	stufenlos	nur ganzzahlige Pixelschritte
Präzision	immer hochpräzise	immer mit Rundungsfehlern
Richtung	Vorwärtstransformation von `p0` nach `p1`: transformiere jeden Vertex aus `p0` nach `p1`	Rückwärtstransformation von `bmp1` nach `bmp0`: suche für jedes Zielpixel von `bmp1` ein Pixel aus `bmp0`
Rand	es gibt keinen Bildrand	großes Problem: Kappung am Bildrand
rückgängig	vollständig reversibel	Operationen fast nie rückgängig zu machen
stapelbar	Operationen sind kaskadierbar	Operationen müssen vom Originalbild `bmp0` ausgehen
überschreiben	`p0` darf von `p1` überschrieben werden	`bmp0` wird meistens noch gebraucht, nicht durch `bmp1` überschreiben!

F: **Scroll eines Rasterbildes um `float dx, float dy` (Input: `bmp0` → Output: `bmp1`)**

```
int idx = Convert.ToInt32( dx );
int idy = Convert.ToInt32( dy );
for ( int y1 = ............................ )
{ int y0 = ..........
  if ( y0 < 0 || y0 >= ySize ) continue;
  for ( int x1 = ............................ )
  { int x0 = ............
    if ( x0 < 0 || x0 >= xSize ) continue;
    Color color = bmp0.GetPixel( ...... );
    bmp1.SetPixel( .................... );
  }
}
A:
int idx = Convert.ToInt32( dx );
int idy = Convert.ToInt32( dy );
for ( int y1 = 0; y1 < bmp1.Height; y1++ )
{ int y0 = y1 - idy;
  if ( y0 < 0 || y0 >= ySize ) continue;
  for ( int x1 = 0; x1 < bmp1.Width; x1++ )
  { int x0 = x1 - idx;
    if ( x0 < 0 || x0 >= xSize ) continue;
    Color color = bmp0.GetPixel( x0, y0 );
    bmp1.SetPixel( x1, y1, color );
  }
}
```

F: **Zoom eines Rasterbildes mit `zoomx`, `zoomy` mit dem Zentrum im Ursprung (Input: `bmp0` → Output: `bmp1`)**

```
for ( int y1 = ............................ )
{ int y0 = Convert.ToInt32( .......... );
  if ( y0 < 0 || y0 >= ySize ) continue;
  for ( int x1 = ............................ )
  { int x0 = Convert.ToInt32( .......... );
    if ( x0 < 0 || x0 >= xSize ) continue;
    Color color = bmp0.GetPixel( ...... );
    bmp1.SetPixel( .................... );
  }
}
A:
for ( int y1 = 0; y1 < bmp1.Height; y1++ )
{ int y0 = Convert.ToInt32( y1 / zoomy );
  if ( y0 < 0 || y0 >= ySize ) continue;
  for ( int x1 = 0; x1 < bmp1.Width; x1++ )
  { int x0 = Convert.ToInt32( x1 / zoomx );
    if ( x0 < 0 || x0 >= xSize ) continue;
    Color color = bmp0.GetPixel( x0, y0 );
    bmp1.SetPixel( x1, y1, color );
  }
}
```

F: **Rotation eines Rasterbildes um `alpha` Grad um den Ursprung im Uhrzeigersinn (Input: `bmp0` → Output: `bmp1`)**

```
double arcus  = ..............................
float sinus   = (float).......................
float cosinus = (float).......................
for ( int y1 = ............................ )
{ float y1_sinus   = y1 * sinus;
  float y1_cosinus = y1 * cosinus;
  for ( int x1 = ........................... )
  { int x0 = Convert.ToInt32( ............................... );
    if ( x0 < 0 || x0 >= xSize ) continue;
    int y0 = Convert.ToInt32( ............................... );
    if ( y0 < 0 || y0 >= ySize ) continue;
    Color color = bmp0.GetPixel( ...... );
    bmp1.SetPixel( ..................... );
  }
}
A:
double arcus  = alpha * 2 * Math.PI / 360;
float sinus   = (float)Math.Sin( arcus );
float cosinus = (float)Math.Cos( arcus );
for ( int y1 = 0; y1 < bmp1.Height; y1++ )
{ float y1_sinus   = y1 * sinus;
  float y1_cosinus = y1 * cosinus;
  for ( int x1 = 0; x1 < bmp1.Width; x1++ )
  { int x0 = Convert.ToInt32( x1 * cosinus + y1_sinus );
    if ( x0 < 0 || x0 >= xSize ) continue;
    int y0 = Convert.ToInt32( -x1 * sinus + y1_cosinus );
    if ( y0 < 0 || y0 >= ySize ) continue;
    Color color = bmp0.GetPixel( x0, y0 );
    bmp1.SetPixel( x1, y1, color );
  }
}
```