

Računarska grafika
(2OER7O02)

Olovke, četke i crtanje primitiva

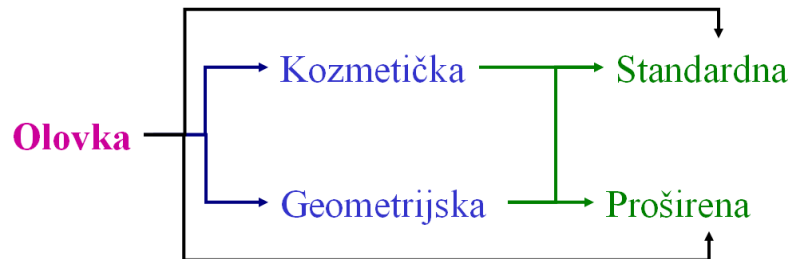
Vežbe



Olovka (*Pen*)

Tipovi olovaka:

- **kozmetička** olovka (*cosmetic*) i
- **geometrijska** olovka (*geometric*)



Kozmetička olovka koristi se kada se zahteva fiksna debljina (1 pixel) i brzo iscrtavanje.

Geometrijska olovka koristi se kada postoji skaliranje linije, definisani spojevi (*joints*), debljina veća od jednog piksela...

Kozmetička olovka

- Dimenzije se definišu u fizičkim jedinicama (*device units*)
- Uvek imaju fiksnu debljinu (1 piksel).
- Linije se iscrtavaju 3-7 puta brže nego sa geometrijskom olovkom.
- Kozmetičke i geometrijske olovke se kreiraju istim naredbama
 - ▷ Atribut debljine se mora postaviti na 0, da bi se kreirala kozmetička olovka.
- Atributi standardne olovke su:
 - ▷ debljina,
 - ▷ stil i
 - ▷ boja.

`CPen::CPen(int nPenStyle, int nWidth, COLORREF crColor);`

Stil kozmetičke olovke

Value	Illustration	Description
PS_SOLID	—————	Puna linija
PS_DASH	· - - - - -	Linija sastavljena od crtica
PS_DOT	Linija sastavljena od tačaka (piksela)
PS_DASHDOT	· - - - - -	Linija sastavljena od kombinacije crtice pa tačka
PS_DASHDOTDOT	— · · - - - -	Linija sastavljena od kombinacije crtice pa dve tačke
PS_NULL		Nevidljiva linija
PS_INSIDEFRAME	—————	Puna linija vidljiva samo unutar zatvorenog oblika

Vrednosti parametra *nPenStyle*

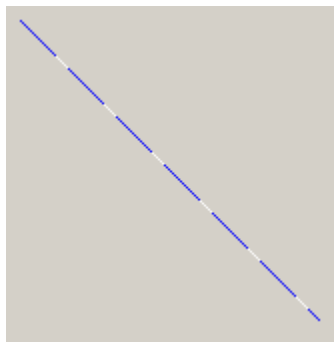
Debljina i boja olovke

- Parametar ***nWidth*** – debljina olovke u logičkim jedinicama
 - ▷ Ako je *nWidth* 0, olovka je debljine jednog piksela bez obzira na transformaciju (kozmetička).
 - ▷ Debljina se prihvata se jedino ukoliko je stil olovke PS_SOLID.
 - ▷ Ako se koristi neki sledećih stilova: PS_DASH, PS_DOT, PS_DASHDOT, PS_DASHDOTDOT, sistem kreira olovku sa stilom PS_SOLID date debljine.
 - Parametar ***crColor*** – boja olovke
- CPen newPen(PS_DASH, 1, RGB(0,0,255));**

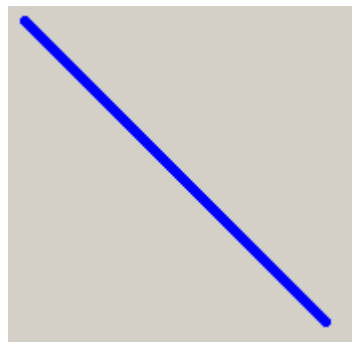
Zanemarivanje stila kod geometrijske olovke

a) `CPen newPen(PS_DASH, 1, RGB(0,0,255));`

b) `CPen newPen(PS_DASH, 5, RGB(0,0,255));`



a)



b)

Geometrijska olovka

- Dimenzije se zadaju u logičkim jedinicama.
- Linije crtane ovakvom olovkom mogu se skalirati (biti deblje ili tanje), zavisno od tekuće *world*-transformacije.
- Atributi proširene geometrijske olovke su:
 - ▷ **debljina, stil i boja (kao i kod kozmetičke),**
 - ▷ **šablon (*pattern*),**
 - ▷ **šrafura (*hatch*),**
 - ▷ **stil kraja (*end style*) i**
 - ▷ **stil spoja (*joint style*).**

Indirektno kreiranje standardnih olovaka

- Parametri standardne olovke mogu se zadati LOGPEN strukturom

```
typedef struct tagLOGPEN {  
    UINT lopnStyle; // PS_SOLID, PS_DASH ...  
    POINT lopnWidth; // x – debljina, y – ne koristi se  
    COLORREF lopnColor;  
} LOGPEN;
```

- A zatim kreirati olovka funkcijom CreatePenIndirect

```
BOOL CPen::CreatePenIndirect( LPLOGPEN lpLogPen );
```


Indirektno kreiranje standardnih olovaka

Parametri proširene olovke definišu se EXTLOGPEN strukturom

```
typedef struct tagEXTLOGPEN {  
    DWORD        elpPenStyle;  
    DWORD        elpWidth;  
    UINT         elpBrushStyle;  
    COLORREF      elpColor;  
    ULONG_PTR     elpHatch;  
    DWORD        elpNumEntries;  
    DWORD        elpStyleEntry[1];  
} EXTLOGPEN;
```

Ali ne postoji funkcija koja kreira olovku na osnovu ove strukture!!!

EXTLOGPEN samo prihvata ono što vraća **GetObject** funkcija.

Kreiranje proširene olovke

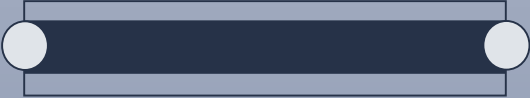

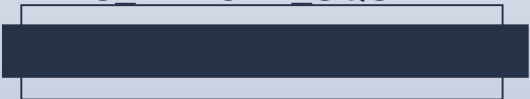

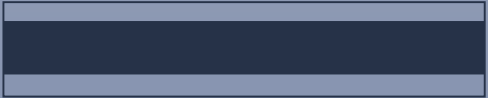

Parametri proširene olovke definišu se EXTLOGPEN strukturom

```
CPen::CPen(  
    int nPenStyle, // PS_GEOMETRIC, PS_COSMETIC , ...  
    int nWidth, // pen width  
    const LOGBRUSH *pLogBrush, // pointer to structure for brush attributes  
    int nStyleCount= 0, // length of array containing custom style bits  
    const DWORD* lpStyle // optional array of custom style bits  
);
```

nPenStyle sadrži:

- ▷ tip (PS_GEOMETRIC, PS_COSMETIC)
- ▷ stil (PS_ALTERNATE, PS_SOLID, PS_DASH, ..., **PS_USERSTYLE**, PS_INSIDEFRAME)
- ▷ završetak (PS_ENDCAP_ROUND, PS_ENDCAP_SQUARE, PS_ENDCAP_FLAT)
- ▷ spoj (PS_JOIN_BEVEL, PS_JOIN_MITER, PS_JOIN_ROUND)

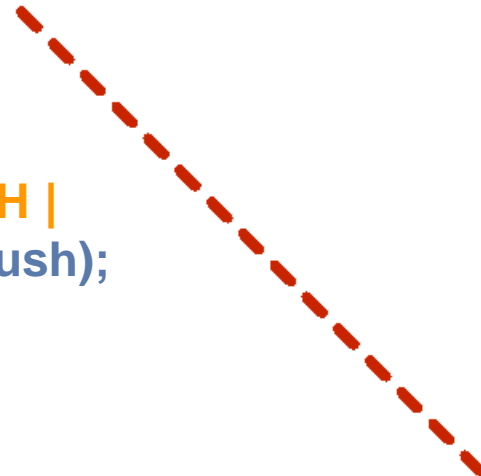
Stilovi proširene olovke

Stil završetka	Stil spoja
<p>PS_ENDCAP_ROUND</p> 	<p>PS_JOIN_BEVEL</p> 
<p>PS_ENDCAP_SQUARE</p> 	<p>PS_JOIN_MITER</p> 
<p>PS_ENDCAP_FLAT</p> 	<p>PS_JOIN_ROUND</p> 

Kreiranje proširene olovke

Primer

```
LOGBRUSH logBrush;  
logBrush.lbStyle = BS_SOLID;  
logBrush.lbColor = RGB(200, 36, 0);  
CPen* pPen = new CPen(PS_GEOMETRIC | PS_DASH |  
PS_ENDCAP_ROUND | PS_JOIN_ROUND, 7, &logBrush);  
CPen* oldPen = pDC->SelectObject(pPen);  
pDC->MoveTo(10, 10);  
pDC->LineTo(300, 300);  
pDC->SelectObject(oldPen);  
delete pPen;
```



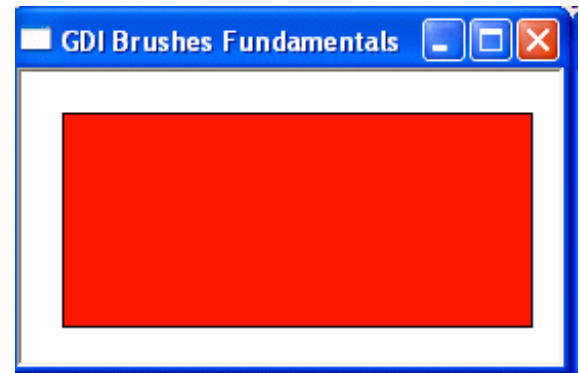
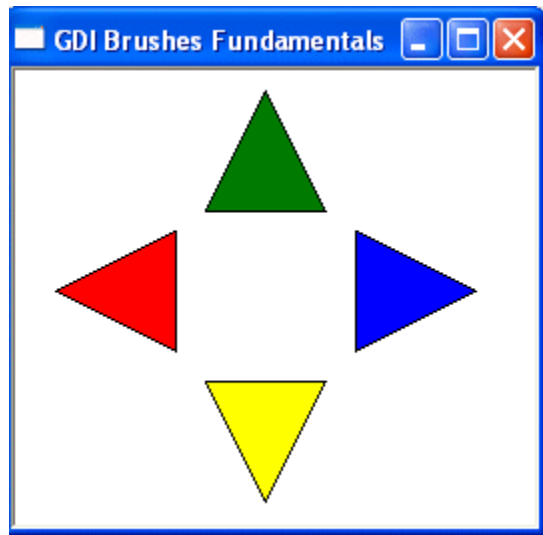
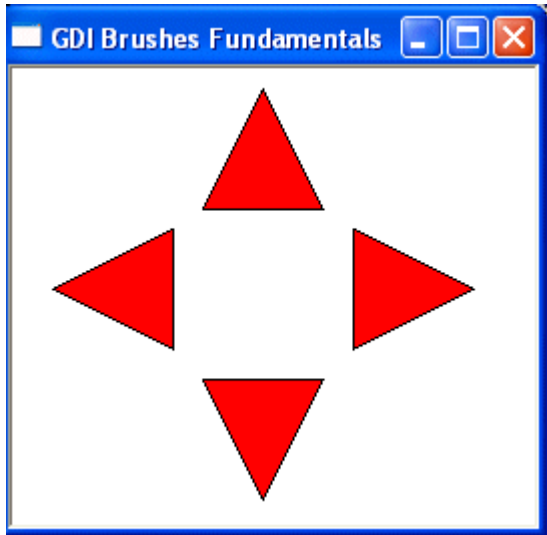
Četka

Četka je grafički objekat koji se koriste za ispunu unutrašnjosti figura kao što su pravougaonik, poligon, elipsa i putanja.

Solid 	iHatch Hatch 
Pattern 	Stock 

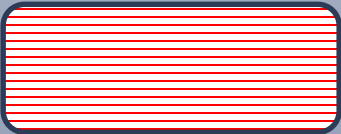
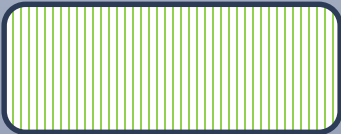
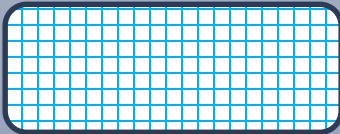
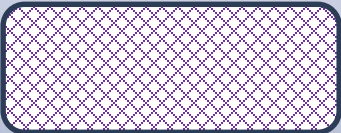
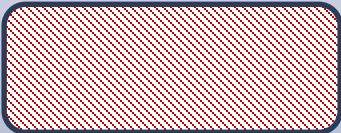

Puna četka (Solid Brush)

`CBrush::CBrush(COLORREF crColor);`



Četka sa šrafurom (Hatched Brush)

`CBrush::CBrush(int nIndex, COLORREF crColor);`

HS_HORIZONTAL 	HS_VERTICAL 	HS_CROSS 
HS_DIAGCROSS 	HS_FDIAGONAL 	HS_BDIAGONAL 

Četka sa bitmapom (Pattern Brush)

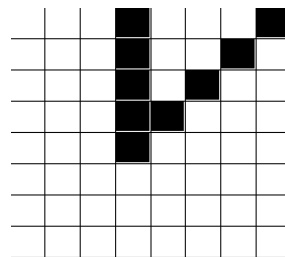
■ Ispuna definisana internom bitmapom – DDB (Device Dependent Bitmap)

- ▷ `CBrush(CBitmap* pBitmap);`
- ▷ `BOOL CBrush::CreatePatternBrush(CBitmap* pBitmap);`

■ Ispuna definisana eksternom bitmapom – DIB-om (Device Independent Bitmap)

- ▷ `BOOL CBrush::CreateDIBPatternBrush(HGLOBAL hPackedDIB, UINT nUsage);`
- ▷ `BOOL CBrush::CreateDIBPatternBrush(const void* lpPackedDIB, UINT nUsage);`

Četka sa bitmapom (Pattern Brush) Primer



```
WORD wBits[8] = { 0xee,0xed,0xeb,0xe7,0xef,0xff,0xff,0xff };
```

```
CBitmap brushBmp;
```

```
brushBmp.CreateBitmap(8, 8, 1, 1, wBits);
```

```
CBrush brush;
```

```
brush.CreatePatternBrush(&brushBmp);
```

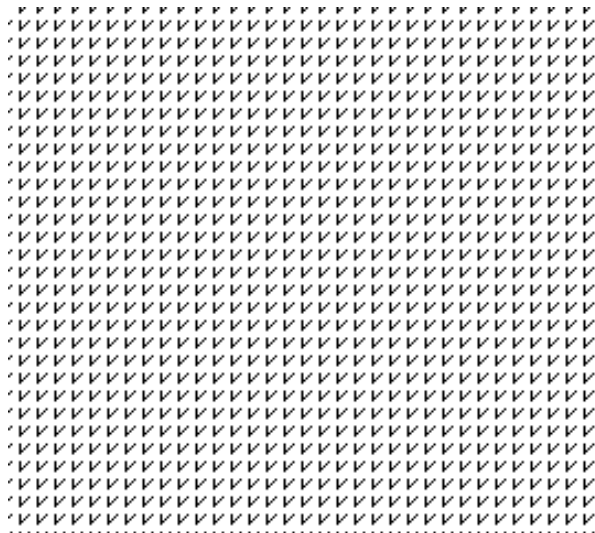
```
CBrush* pOldBrush = pDC->SelectObject(&brush);
```

```
pDC->Rectangle(0, 0, 500, 500);
```

```
pDC->SelectObject(pOldBrush);
```

```
brushBmp.DeleteObject();
```

```
brush.DeleteObject();
```



Indirektno kreiranje četke

Parametri svih vrsta četki mogu se zadati LOGBRUSH strukturom

```
typedef struct tagLOGBRUSH {  
    UINT lbStyle; // BS_SOLID, BS_PATTERN, BS_HATCHED, ...  
    COLORREF lbColor; // DIB_PAL_COLORS, DIB_RGB_COLORS  
    LONG lbHatch; // HS_BDIAGONAL, HS_CROSS, ...  
} LOGBRUSH;
```

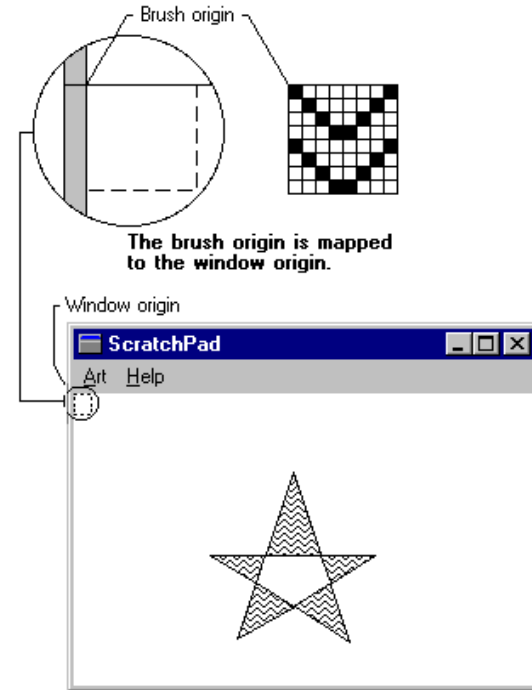
- ▶ Ako je stil četke **BS_PATTERN**, *lbHatch* je hendl bitmape
- ▶ Ako je stil četke **BS_SOLID** ili **BS_HOLLOW**, *lbHatch* se ignoriše.

Četka se kreira metodom CreateBrushIndirect

```
BOOL CBrush::CreateBrushIndirect(const LOGBRUSH*  
    lpLogBrush );
```

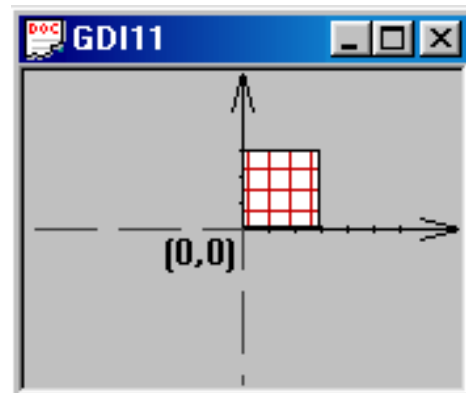
Početak četke (origin)

- Piksel (0,0) bitmape koja predstavlja četku se postavlja u koordinatni početak prozora.
 - Ceo prozor se “ispunjava” izabranom bitmapom, pri čemu je ispuna vidljiva samo u unutrašnjosti figura koje se crtaju.
- `CPoint CDC::SetBrushOrg(int x, int y);`
`CPoint CDC::GetBrushOrg();`
- x i y su koordinate (uređaja) u intervalu [0,7] koje predstavljaju novi početak za iscrtavanje bitmapa (šrafure) za ispunu
 - Početak četke se pimenjuje na sledeću četku koja će biti selektovana u kontekst uređaja.



Četka sa bitmapom (Pattern Brush) Primer

```
CBrush brush;  
brush.CreateHatchBrush(HS_CROSS,RGB(200,0,0));  
brush.UnrealizeObject();  
pDC->SetBrushOrg(1,1);  
CBrush* pOldBrush=pDC->SelectObject(&brush);  
pDC->Rectangle(CRect(0,0,30,30));  
pDC->SelectObject(pOldBrush);  
brush.DeleteObject();
```



Gotovi GDI objekti

virtual CGdiObject*
CDC::SelectStockObject
(int nIndex)

- BLACK_PEN
- NULL_PEN
- WHITE_PEN

- DKGRAY_BRUSH
- GRAY_BRUSH
- HOLLOW_BRUSH
- LTGRAY_BRUSH
- NULL_BRUSH
- WHITE_BRUSH

Crtanje tačke

Vrednost piksela u tački sa zadatim koordinatama se pribavlja/postavlja na zadatu boju

```
COLORREF CDC::GetPixel( int x, int y ) const;
```

```
COLORREF CDC::GetPixel( POINT point ) const;
```

```
COLORREF CDC::SetPixel( int x, int y, COLORREF crColor );
```

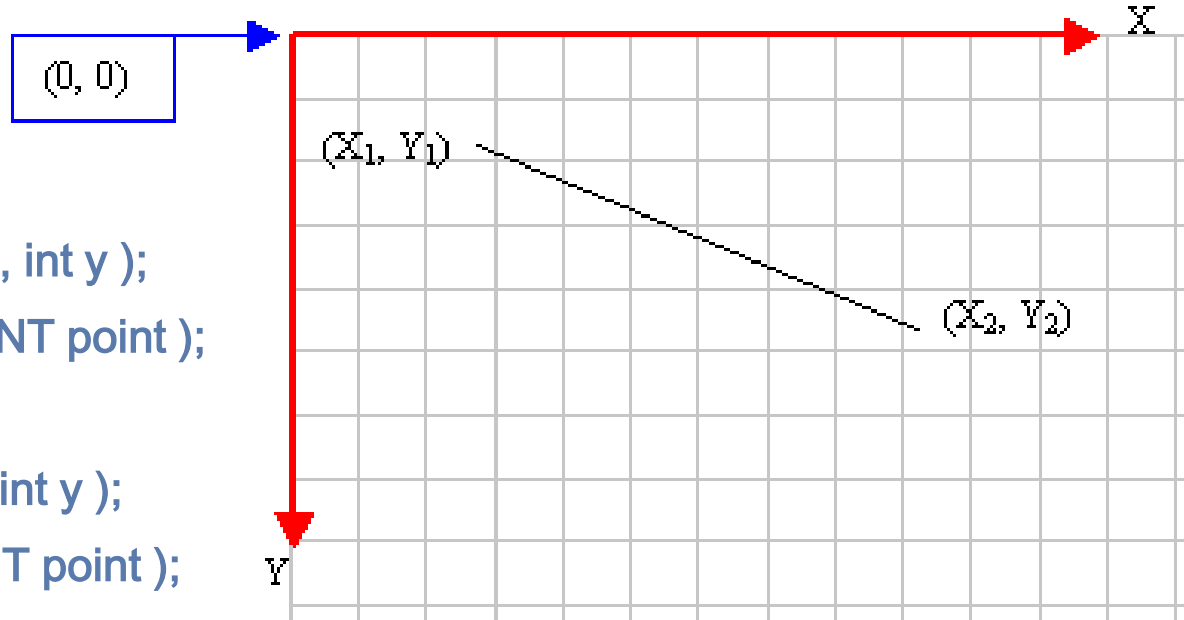
```
COLORREF CDC::SetPixel( POINT point, COLORREF crColor );
```

Parametri metoda

- ▷ **x** – logička x-koordinata tačke
- ▷ **y** – logička y-koordinata tačke
- ▷ **point** – (x,y) koordinate su definisane strukturom POINT ili klasom CPoint
- ▷ **crColor** – boja na koju se postavlja piksel

Crtanje linije

```
CPoint CDC::MoveTo( int x, int y );  
CPoint CDC::MoveTo( POINT point );  
  
CPoint CDC::LineTo( int x, int y );  
CPoint CDC::LineTo( POINT point );
```



Crtanje izlomljene linije

■ Crtanje jedne izlomljene linije

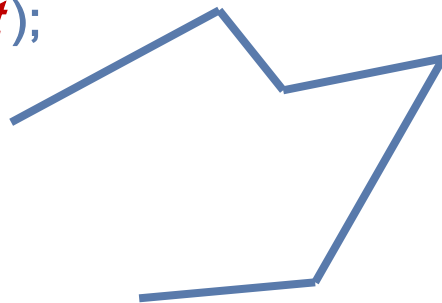
```
BOOL CDC::Polyline( LPPOINT lpPoints, int nCount );
```

- ▷ *lpPoints* – niz temena izlomljene linije
- ▷ *nCount* – broj temena izlomljene linije

■ Crtanje niza izlomljenih linija

```
BOOL CDC::PolyPolyline( const POINT* lpPoints,  
const DWORD* lpPolyPoints, int nCount );
```

- ▷ *lpPoints* – niz temena svih izlomljenih linija
- ▷ *lpPolyPoints* – niz brojeva, gde je svaki element broj tačaka jedne izlomljene linije
- ▷ *nCount* – broj izlomljenih linija (mora biti najmanje 2)



Crtanje mnogougla

Crtanje jednog mnogougla

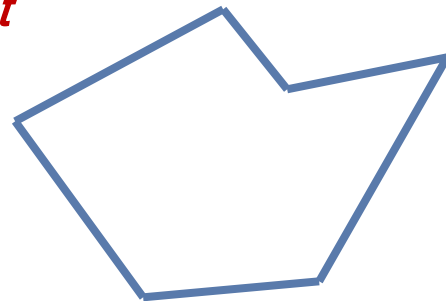
```
BOOL CDC::Polygon( LPPOINT lpPoints, int nCount );
```

- *lpPoints* – niz temena mnogougla
- *nCount* – broj temena mnogougla

Crtanje niza mnogouglova

```
BOOL CDC::PolyPolygon( const POINT* lpPoints,  
const DWORD* lpPolyPoints, int nCount );
```

- *lpPoints* – niz temena svih mnogouglova
- *lpPolyPoints* – niz brojeva, gde je svaki element broj tačaka jednog mnogougla
- *nCount* – broj mnogouglova (mora biti najmanje 2)



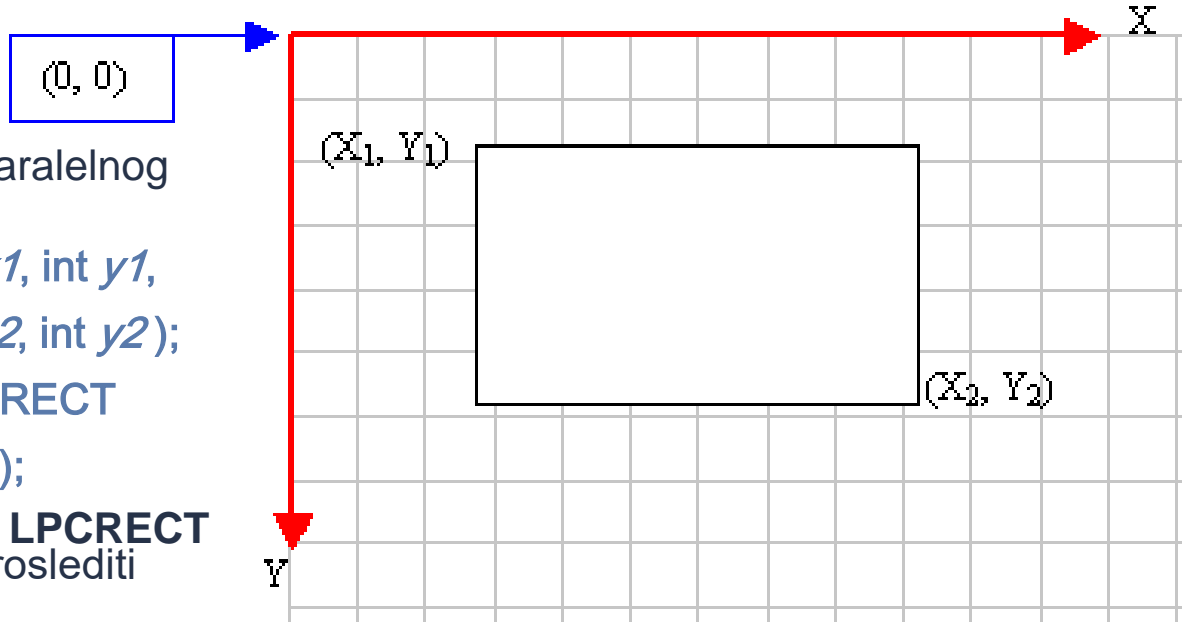
Crtanje pravougaonika

- Crtanje pravougaonika paralelnog koordinatnim osama

```
BOOL CDC::Rectangle( int x1, int y1,  
                    int x2, int y2);
```

```
BOOL CDC::Rectangle( LPCRECT  
                    lpRect);
```

- Parametar **lpRect** je tipa **LPCRECT** na čije mesto se može proslediti argument tipa **CRect**



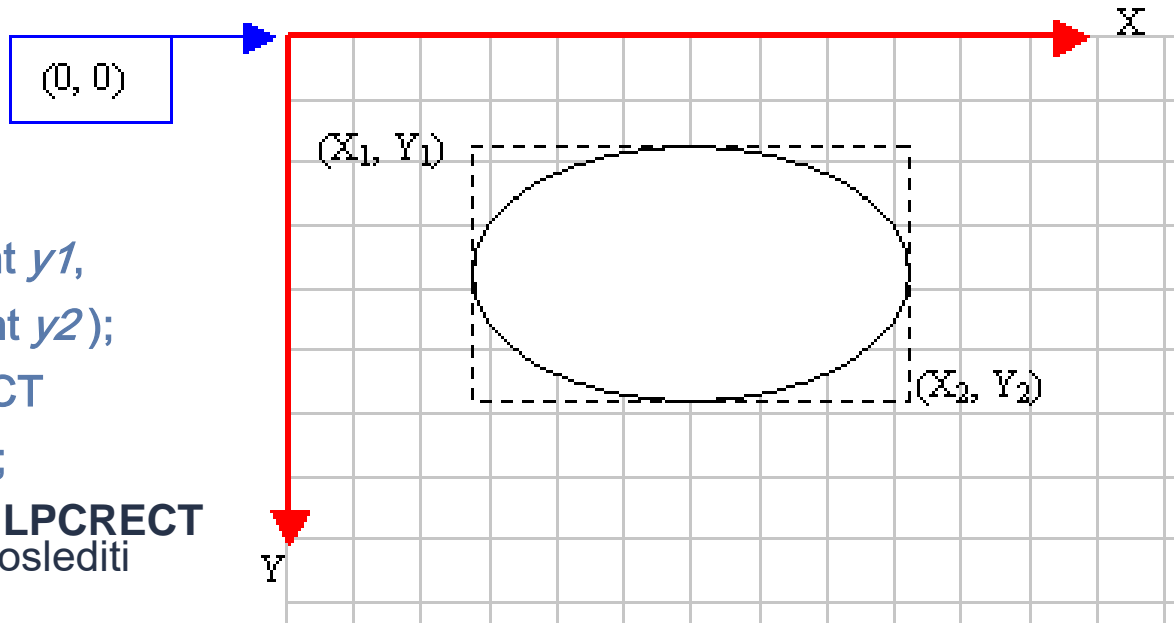
Crtanje elipse

- Crtanje elipse paralelne koordinatnim osama

```
BOOL CDC::Ellipse( int x1, int y1,  
                  int x2, int y2);
```

```
BOOL CDC::Ellipse( LPCRECT  
                  lpRect);
```

- Parametar **lpRect** je tipa **LPCRECT** na čije mesto se može proslediti argument tipa **CRect**



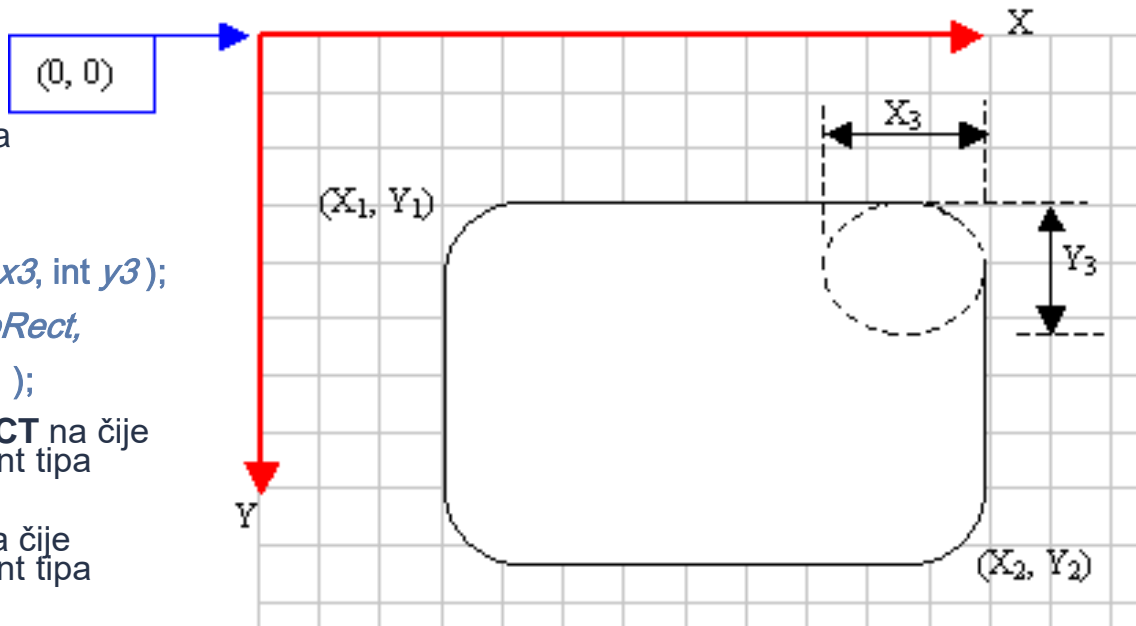
Crtanje zaobljenog pravougaonika

- Crtanje zaobljenog pravougaonika paralelnog koordinatnim osama

```
BOOL CDC::RoundRect( int x1, int y1,  
                    int x2, int y2, int x3, int y3);
```

```
BOOL CDC::RoundRect( LPCRECT lpRect,  
                    POINT point );
```

- Parametar ***lpRect*** je tipa **LPCRECT** na čije mesto se može proslediti argument tipa **CRect**
- Parametar ***point*** je tipa **POINT** na čije mesto se može proslediti argument tipa **CPoint**



Crtanje luka

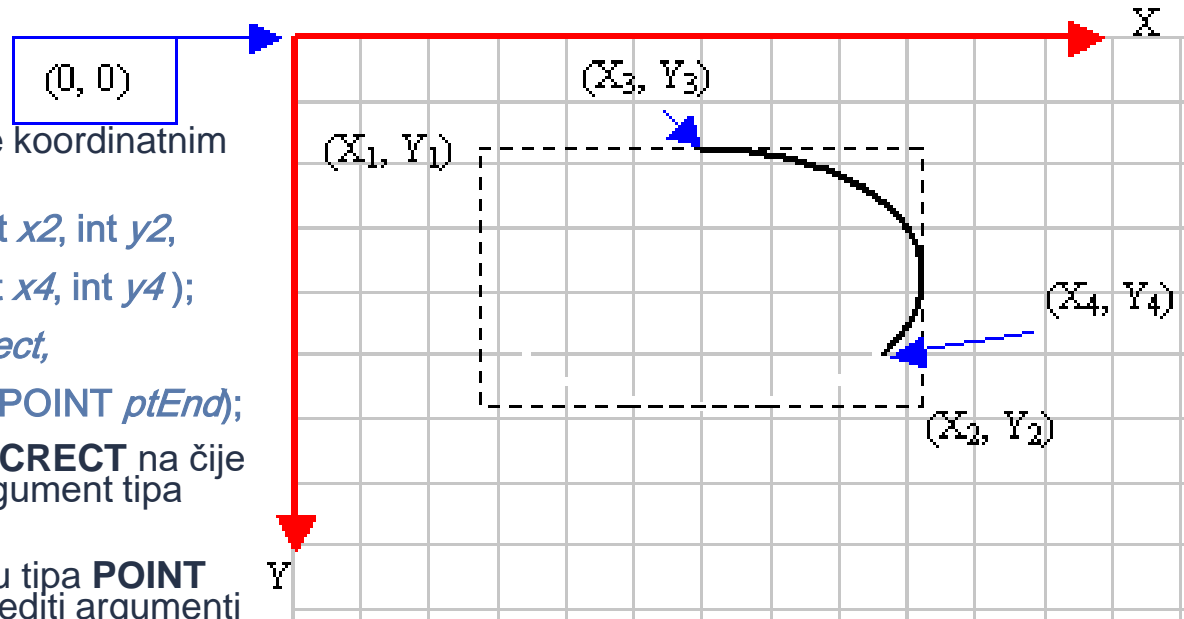
Crtanje luka elipse paralelne koordinatnim osama

```
BOOL CDC::Arc( int x1, int y1, int x2, int y2,  
               int x3, int y3, int x4, int y4 );
```

```
BOOL CDC::Arc( LPCRECT lpRect,  
               POINT ptStart, POINT ptEnd );
```

Parametar **lpRect** je tipa **LPCRECT** na čije mesto se može proslediti argument tipa **CRect**

Parametri **ptStart** i **ptEnd** su tipa **POINT** na čije mesto se mogu proslediti argumenti tipa **CPoint**



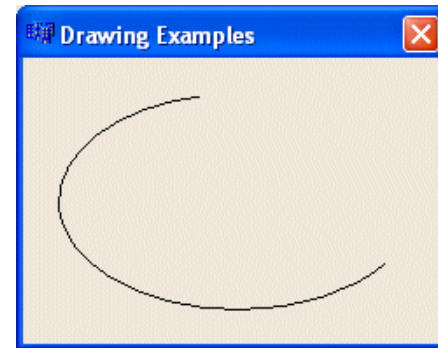
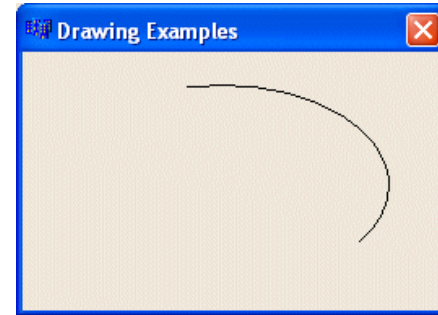
Orientacija luka

- Orientacija luka se pribavlja/postavlja na zadatu vrednost

```
int CDC::GetArcDirection( );
```

```
int CDC::SetArcDirection( int nArcDirection );
```

- ***nArcDirection*** – orientacija luka može imati dve vrednosti:
 - ▶ **AD_COUNTERCLOCKWISE** – orientacija suprotna od smeru kretanja kazaljke na časovniku
 - ▶ **AD_CLOCKWISE** – orientacija u smeru kretanja kazaljke na časovniku
- Podrazumevana orientacija je suprotna od smeru kretanja kazaljke na časovniku



Crtanje pite

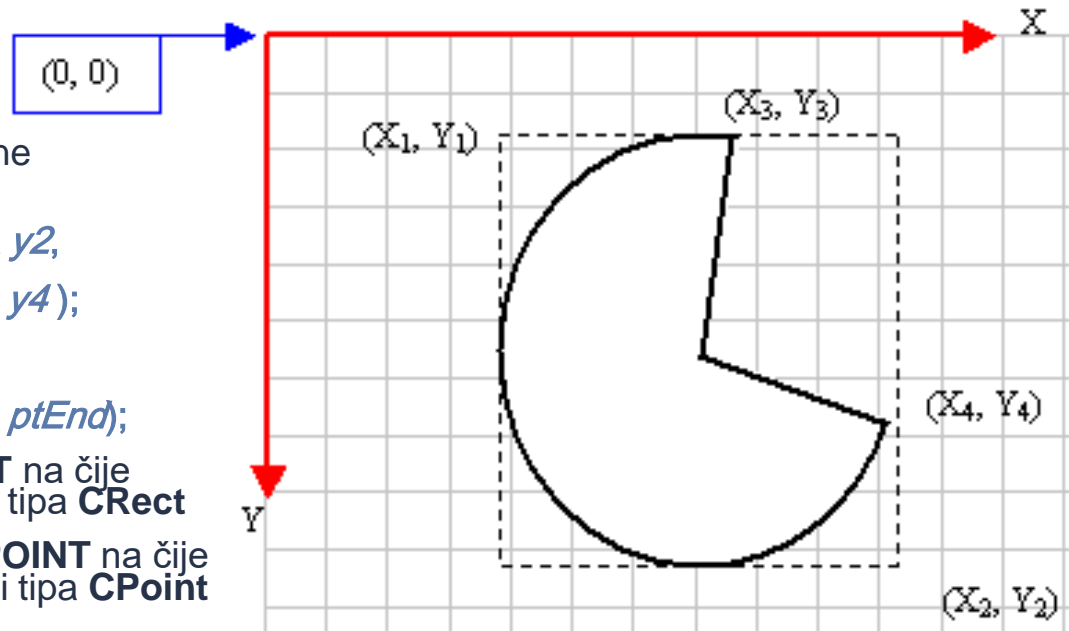
Crtanje pite kao dela elipse paralelne koordinatnim osama

```
BOOL CDC::Pie( int x1, int y1, int x2, int y2,  
               int x3, int y3, int x4, int y4 );
```

```
BOOL CDC::Pie( LPCRECT lpRect,  
               POINT ptStart, POINT ptEnd );
```

Parametar **lpRect** je tipa **LPCRECT** na čije mesto se može proslediti argument tipa **CRect**

Parametri **ptStart** i **ptEnd** su tipa **POINT** na čije mesto se mogu proslediti argumenti tipa **CPoint**



Crtanje odsečka

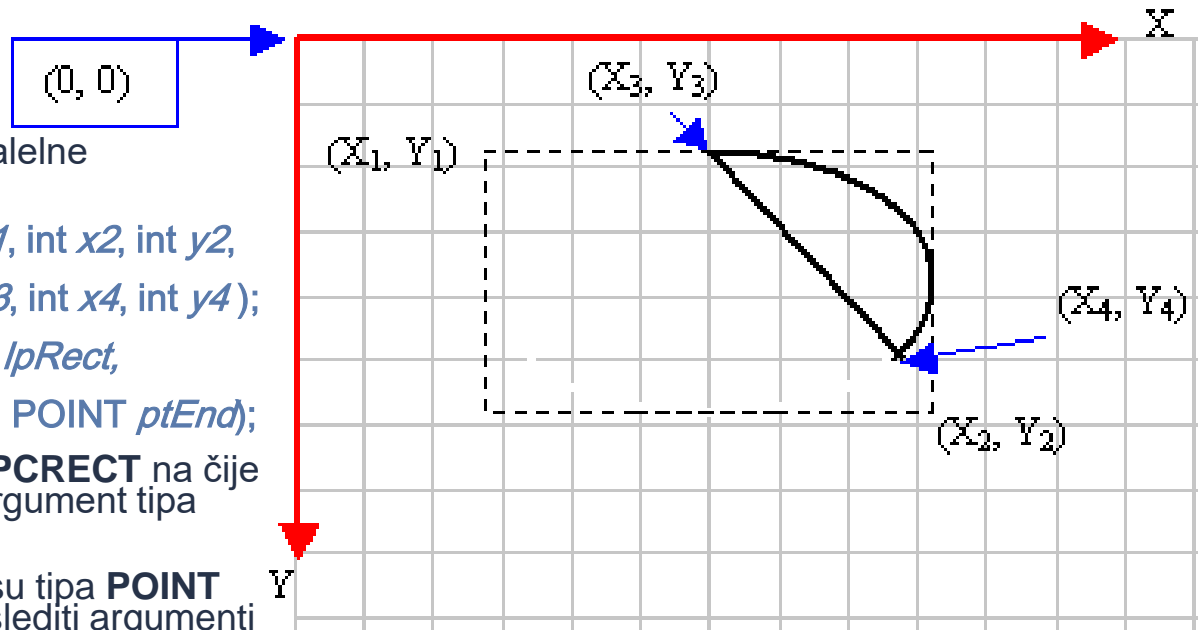
- Crtanje odsečka elipse paralelne koordinatnim osama

```
BOOL CDC::Chord( int x1, int y1, int x2, int y2,  
                 int x3, int y3, int x4, int y4 );
```

```
BOOL CDC:: Chord( LPCRECT lpRect,  
                 POINT ptStart, POINT ptEnd);
```

- Parametar **lpRect** je tipa **LPCRECT** na čije mesto se može proslediti argument tipa **CRect**

- Parametri **ptStart** i **ptEnd** su tipa **POINT** na čije mesto se mogu proslediti argumenti tipa **CPoint**



Crtanje Bezier-ove krive

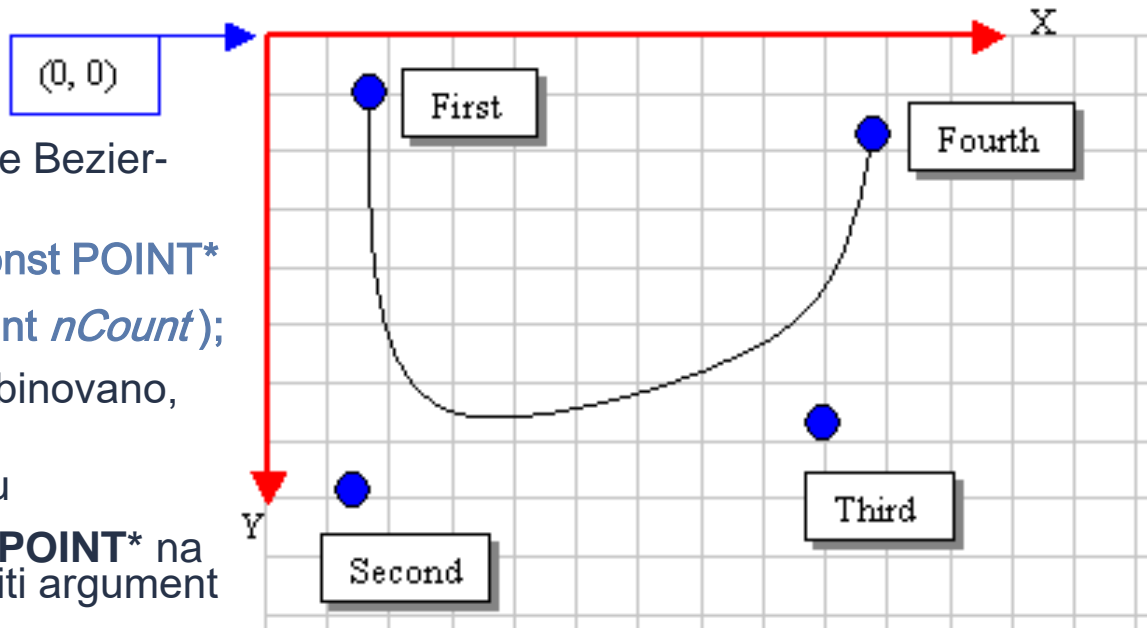
Crtanje krive linije definisane Bezier-ovom krivom

```
BOOL CDC::PolyBezier( const POINT*  
                      lpPoints, int nCount);
```

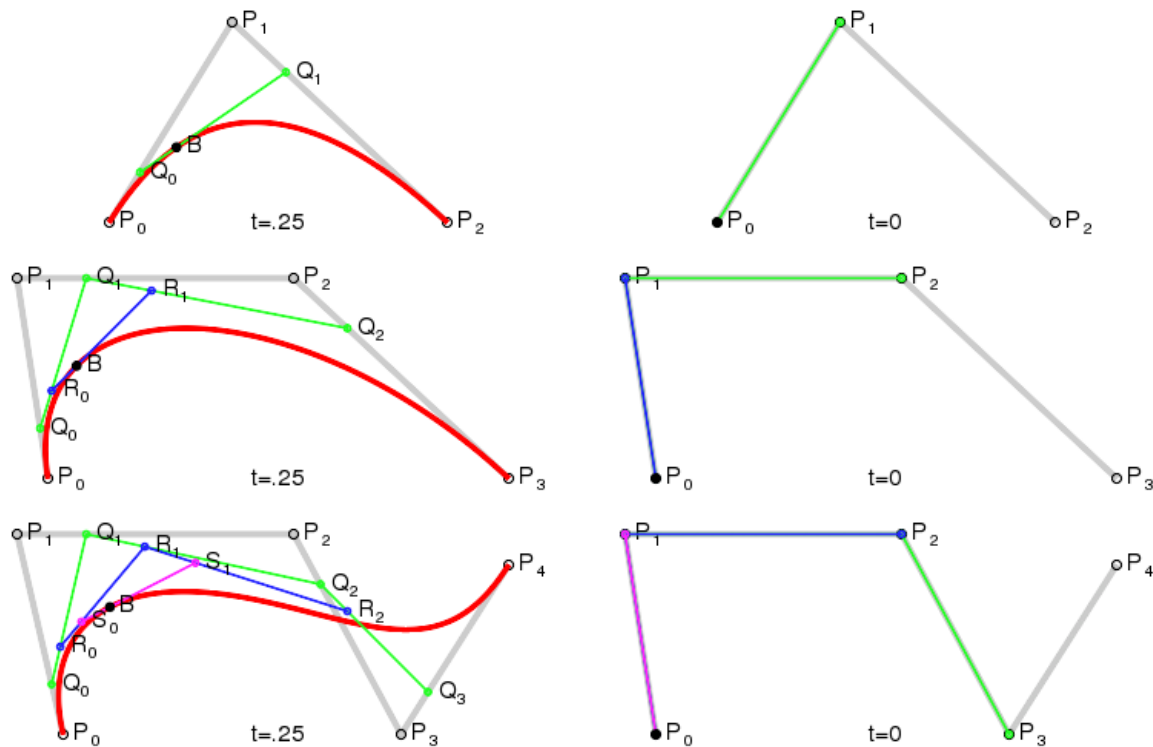
lpPoints – niz tačaka (kombinovano, krajnje i kontrolne)

nCount – broj tačaka u nizu

Parametar **lpPoints** je tipa **POINT*** na čije mesto se može proslediti argument tipa **CPoint***



Bezier-ova kriva



Bezier-ova kriva

- Bezier-ova kriva može se predstaviti u parametarskom obliku:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + x_0$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + y_0$$

- Ukoliko su poznati koeficijenti a_x , b_x , c_x i početna tačka $T_0(x_0, y_0)$, tačke krive $T_1(x_1, y_1)$, $T_2(x_2, y_2)$ i $T_3(x_3, y_3)$ se dobijaju rešavanjem sledećih jednačina:

$$x_1 = x_0 + c_x / 3$$

$$x_2 = x_1 + (c_x + b_x) / 3$$

$$x_3 = x_0 + a_x + b_x + c_x$$

Bezier-ova kriva

- Bezier-ova kriva može se predstaviti u parametarskom obliku:

$$\mathbf{x}(t) = \mathbf{a}_x t^3 + \mathbf{b}_x t^2 + \mathbf{c}_x t + \mathbf{x}_0$$

$$\mathbf{y}(t) = \mathbf{a}_y t^3 + \mathbf{b}_y t^2 + \mathbf{c}_y t + \mathbf{y}_0$$

- Ukoliko su poznate tačke $T_0(x_0, y_0)$, $T_1(x_1, y_1)$, $T_2(x_2, y_2)$ i $T_3(x_3, y_3)$, koeficijenti \mathbf{a}_x , \mathbf{b}_x i \mathbf{c}_x se dobijaju rešavanjem sledećih jednačina:

$$\mathbf{c}_x = 3(\mathbf{x}_1 - \mathbf{x}_0)$$

$$\mathbf{b}_x = 3(\mathbf{x}_2 - 2\mathbf{x}_1 + \mathbf{x}_0)$$

$$\mathbf{a}_x = \mathbf{x}_3 + 3(\mathbf{x}_1 - \mathbf{x}_2) - \mathbf{x}_0$$

GDI+

Olovka

- Objekat Pen može se kreirati na 2 načina

```
Pen( const Brush* brush, REAL width );
```

```
Pen( const Color& color, REAL width );
```

- Primer

```
Pen pen(Color(255,255,0,0), 4.0f);
```

```
graphics.DrawLine(&pen, 0, 0, 200, 100);
```

Definisanje krajeva linija

```
Pen pen(Color(255, 0, 0, 255), 8);  
stat = pen.SetStartCap(LineCapArrowAnchor);  
stat = pen.SetEndCap(LineCapRoundAnchor);  
stat = graphics.DrawLine(&pen, 20, 175, 300, 175);
```

Različite primitive za crtanje linija

`DrawArc(Pen* pen, Rect& rect, REAL startAngle, REAL sweepAngle)`

`DrawBezier(Pen* pen, Point& pt1, Point& pt2, Point& pt3, Point& pt4)`

`DrawBeziers(Pen* pen, Point* points, INT count)`

`DrawClosedCurve(Pen* pen, Point* points, INT count, REAL tension)`

`DrawCurve(Pen* pen, Point* points, INT count, INT offset, INT numberOfSegments, REAL tension)`

`DrawCurve(Pen* pen, Point* points, INT count, REAL tension)`

`DrawEllipse(Pen* pen, Rect& rect)`

`DrawEllipse(Pen* pen, INT x, INT y, INT width, INT height)`

`DrawPath(pen, path)`

`DrawPie(Pen* pen, Rect& rect, REAL startAngle, REAL sweepAngle)`

`DrawPolygon(Pen* pen, Point* points, INT count)`

`DrawRectangle(Pen* pen, Rect& rect)`

Četka

■ Puna četka

`SolidBrush(const Color& color);`

■ Četka sa šrafurom

`HatchBrush(HatchStyle hatchStyle, const Color& foreColor, const Color& backColor);`

- Postoji preko 50 predifinisanih tipova šrafura

■ Četka sa bitmapom

`TextureBrush(Image* image, WrapMode wrapMode);`

- `enum WrapMode { WrapModeTile, WrapModeTileFlipX, WrapModeTileFlipY, WrapModeTileFlipXY, WrapModeClamp };`

Četka Primer

```
Color black(255,0,0,0);  
Color white(255,255,255,255);  
HatchBrush brush(HatchStyleHorizontal, black, white);  
graphics.FillRectangle(&brush, 20, 20, 100, 50);  
HatchBrush brush1(HatchStyleVertical, black, white);  
graphics.FillRectangle(&brush1, 120, 20, 100, 50);  
HatchBrush brush2(HatchStyleForwardDiagonal, black, white);  
graphics.FillRectangle(&brush2, 220, 20, 100, 50);  
HatchBrush brush3(HatchStyleBackwardDiagonal, black, white);  
graphics.FillRectangle(&brush3, 320, 20, 100, 50);
```

Različite primitive za ispunu figura

FillClosedCurve(Brush* brush, Point* points, INT count)

FillEllipse(Brush* brush, Rect& rect)

FillPath(brush, path)

FillPie(Brush* brush, Rect& rect, REAL startAngle, REAL sweepAngle)

FillPolygon(Brush* brush, Point* points, INT count)

FillRegion(brush, region)

QPainter

Promena atributa objekta

- Moguća direktna promenu atributa selektovanih objekata:

```
QPainter p(this);
```

```
p.setPen(QColor(255,0,0,255));
```

- Moguće je kreiranje novih objekata, a zatim njihova selekcija:

```
QPainter p(this);
```

```
QPen pen;
```

```
pen.setColor(QColor(255, 255, 255, 128));
```

```
p.setPen(pen);
```

Olovka

Kreiranje

- Olovka debljine 1 i proizvoljnog stila

`QPen::QPen(Qt::PenStyle style)`

- Olovka debljine 1 i prosleđene boje

`QPen::QPen(const QColor & color)`

- Olovka sa svim parametrima

`QPen::QPen(const QBrush & brush, qreal width, Qt::PenStyle style = Qt::SolidLine, Qt::PenCapStyle cap = Qt::SquareCap, Qt::PenJoinStyle join = Qt::BevelJoin)`

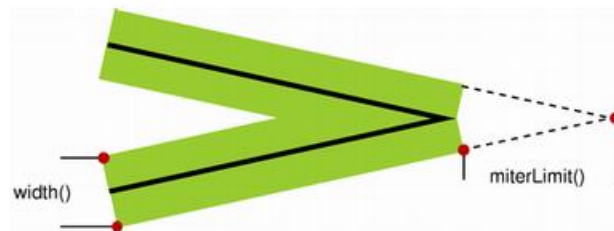
- Kopija druge olovke

`QPen::QPen(const QPen & pen)`

Olovka

Promena atributa

```
void QPen::setBrush(const QBrush & brush)
void QPen::setCapStyle(Qt::PenCapStyle style)
void QPen::setColor(const QColor & color)
void QPen::setCosmetic(bool cosmetic)
void QPen::setDashOffset(qreal offset)
void QPen::setDashPattern(const QVector<qreal> & pattern)
void QPen::setJoinStyle(Qt::PenJoinStyle style)
void QPen::setMiterLimit(qreal limit)
void QPen::setStyle(Qt::PenStyle style)
void QPen::setWidth(int width)
void QPen::setWidthF(qreal width)
```



Četka

- Promena boje četke

`void QBrush::setColor(const QColor & color)`

- Transformacija položaja četke (superponira na transformaciju QPainter-a)

`void QBrush::setMatrix(const QMatrix & matrix)`

`void QBrush::setTransform(const QTransform &matrix)`

- Promena šrafure četke

`void QBrush::setStyle(Qt::BrushStyle style)`

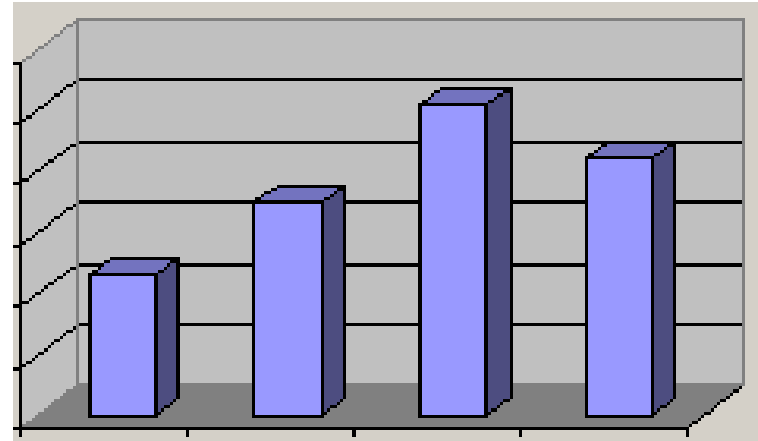
- Promena bitmape četke

`void QBrush::setTexture(const QPixmap & pixmap)`

`void QBrush::setTextureImage(const QImage &image)`

Zadatak za vežbu

Napisati funkciju koja iscrtava 3D dijagram sa stupcima (kao u Excel-u). Ulazni parametri f-je su: okvirni pravougaonik koji definiše gde u prozoru treba iscrtati grafikon (u logičkim koordinatama), vektor realnih brojeva kojim se zadaju vrednosti koje treba predstaviti na grafikonu, dužina prethodnog vektora i boja stubaca (njihove prednje strane). Smatrati da su stupci razmaknuti za 1.5 debljina jednog stupca, i da se njihove visine i debljine menjaju zavisno od zadatog okvirnog pravougaonika.



Pitanja

